DOUBLE ISSUE

Volume 1
Number 2

ON THREE

T.M.

The Magazine For Apple III Owners and Users

$3.00

February-March
1983

- Disk Pak 2: Listing files in Pascal
- Alpha Who?
- Pascal Tutorial begins
- REVIEW ON: Apple Writer ///
- Plus much more!

ON THREE T.M.

The Magazine For Apple III Owners and Users

ON THREE
Post Office Box 3825
Ventura, California 93006
(805)644-3514

Another open letter to our subscribers

We're back again, but we're late! Over the past few months I have had the pleasure to put together the first and only magazine specifically for Apple /// owners and users. It has been a joy for me to work those late nights in front of my trusty old Apple /// and I know by your support that I will be doing this for quite some time.

The delays in this issue were caused by a variety of factors that I would like to forget. In January we mailed out a sample copy of the magazine to all Apple dealerships in the country. As this issue goes to press, a check consisting of spot calls to various dealers around the country reveals that less than 10% of the dealers who responded had received it. So much for bulk mail.

We were counting on the response from that mailing to reach many more thousands of Apple /// users. Because of the snail-paced delivery, we have not yet been able to reach the number of Apple /// users that we need in order to run the quality publication that the /// deserves.

Since we do not have that large a subscription base, we are forced to give you a February-March issue. Also a single April-May issue is planned. For those of you worrying about your yearly subscription fee, please don't. All subscribers will receive 12 issues regardless of whether it takes 14 or 15 months to do it. This issue will count as one, and the upcoming April-May issue will count as one. Come June we hope, plan and pray that we will be able to go to a monthly format.

In your own interests I therefore ask that you give your dealer a call and ask him to stock ON THREE. The quicker we get larger numbers of people reading the magazine, the quicker we will be able to go monthly. Again I thank you for your support.

Sincerely,

Bob Consorti

Bob Consorti
Editor

ON THREE

February-March, 1983
Volume 1 Number 2

Editor/Publisher:
Bob Consorti

Managing Consultant:
Joseph Consorti

Cover Design and Artwork:
William V. Padula
Cranford, New Jersey

Interior Artwork:
Virginia Carol

Typesetting Services:
The Typesetting Company
Van Nuys, California

Printing Services:
Ojai Printing &
Publishing Company
Ojai, California

/// TABLE OF CONTENTS ///

/// FEATURES ///

/// DEPARTMENTS ///

Next Month in ON THREE

DOS file list ... Reset lock & unlock ... Reboot ... Slow-Fast ... Horses ON the *III* ... Reviews: Profile, Backup *III*, Word Processing Language, Quick & Easy Data Master, Critical Path Scheduling ... same old tutorials ... and more!

ON THREE **- THE** Reference Source for the Apple /// is published somewhat monthly by ***ON THREE,*** P.O. Box 3825, Ventura, California 93006.

For a copy of our Author Guidelines, please send a self-addressed stamped envelope (20 cents) to the above address.

Subscription information: U.S. - $30 for 12 issues. For first class mail, please remit an extra $10.

All foreign subscriptions should include additional postage in the following amounts:

$8 for Canada and Mexico
$12 for Central America - Caribbean
$16 for South America - Europe - Africa
$19 for Asia - Pacific Islands - Australia

Funds should be remitted in either U.S. dollars drawn on a U.S. bank, International Money Order, or by a direct bank draft.

Group Rates are as follows:

2 - 9 members, $28 per subscription
10-49 members, $25 per subscription
50 + members, $22 per subscription

Group purchases must have one mailing address.

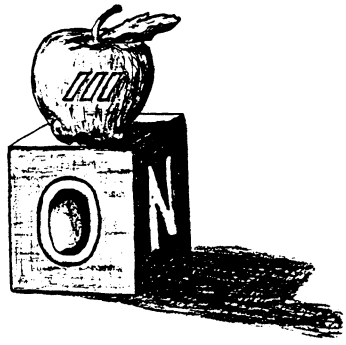
Return postage must accompany all manuscripts, drawings, and diskettes submitted if they are to be returned, and no responsibility can be assumed for unsolicited materials.

All letters sent to ***ON THREE*** will be treated as unconditionally assigned for publication and are subject to ***ON THREE's*** right to edit and comment editorially.

Dealer inquiries are welcomed. Please write to the above address for volume pricing and terms.

The Editors Block:

Bob Consorti



As I sit down to write this column, I can't forget a phone conversation that I had this morning with a dealer. She called to place an order and we began talking. She told me that she and all the Apple /// users in her area know that the Apple /// is one of the finest machines available today.

The conversation then went to the subject of why no one is saying just that. Most people think that the Apple /// is a dog that needs to be put out of its misery. Even Apple can't figure out why the /// is not selling as well as the][. It seems that we are the first people, the first magazine that is trying to remedy this situation.

How did we get in this spot? I can sum it up in one word -Openness -. For the two and one half years the Apple /// has been around, if you wanted to learn the internals of SOS, the heart and soul of the Apple ///, you had to spend a grand or so and attend a special workshop at Cupertino. Two and one half years and the best technical information available is the Beta draft of the SOS Reference Manual.

Apple has been promising that this manual would be available 'soon' since the end of 1980. If you look on your calendar it's 1983! Why is it taking so long? Read the article 'The ///'s For Me' in this issue. As Al Evans says, the explosive growth of the Apple][was the work of the "Crazies"- that strange group of people who need the information that Apple has not been making available.

With the introduction of the IBM Personal Computer, IBM concurrently released all of the system reference manuals. Nothing was held back! Look at where the IBM PC is today! The Apple ///, with powerful SOS is a generation ahead of anything else on the market, yet it is lagging far behind in sales.

If the Apple /// had been 'open' from the start, maybe we wouldn't be in the situation we are in today. For the Apple /// to be the success that it deserves to be, three (ha!) things must happen.

First of all, Apple has to develop a clearer perspective on the problems of the /// and become committed to supporting it better in the future. Part of this means that they should release all technical information to the general population of Apple /// users, and make available any information that those people ask for.

Secondly, this magazine must publish as much of that information as possible to spread the various 'tricks' that your /// can be made to do. The most important part of the magazine aspect is the flow of ideas - the open forum, where people can ask questions - and get answers. Before we published the letter talking about the product by Micro-Sci that allows you to run all Apple][game software, how many people had heard about it? Without this forum, things can not improve.

What is the last thing that must happen for the /// to be a success? You guessed it - People! We have to get to as many Apple /// users as possible. 99% isn't too much, for without those people, new ideas will go nowhere, and we will be back at the start.

Are all these things going to happen? You bet they are! Apple knows that there is a problem, and they aren't as naive a company as some people perceive them to be. They will clean up their act, and start to help the /// become the success that it deserves.

What about us? Can ON THREE do what it needs to do? I most certainly think so. If you look over the first two issues, you will see some very technical articles hidden under the guise of something useful that your Apple /// can do. The article 'Disk Pak1' in the January issue and the two articles 'Disk Pak2' and 'SOS Directory Structure Revealed' in this issue, show the user how to do some very interesting things with the /// while giving some information that was previously unknown.

Since I don't want this magazine to become too 'techie', we are also going to publish as many tutorials and reviews as we can. These are for the normal Apple /// user who wants nothing more than the information on which word processor or data base system is best. We have to be a 'Jack Of All Trades' and please everyone, or we won't be pleasing anyone!

What's left? What needs to be done? That's right! People, we have to get to people. I want every Apple /// owner in the country to know about us so we can provide everyone with the kind of information that can make the Apple /// the biggest hit in the computer field, but I need your help! Tell a friend! Call your computer store and ask them to stock ON THREE. The more people we can reach the bigger the Apple /// will become, so get cracking!

Wow! I think I got a little bit too emotional. Heck, I can't help it - I want the /// to succeed so badly! Remember, you can make a difference - so please try.

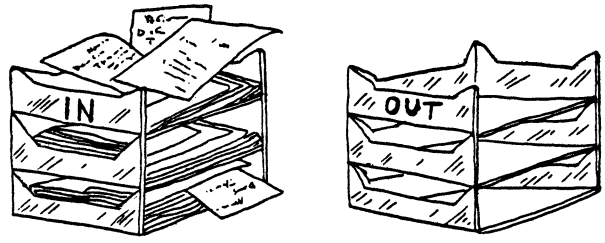
Coming back to earth, let's take a look inside the latest issue of ON THREE! Al Evans joins our ranks as a contributing author this month with the article 'The ///'s For Me', a short dissertation into the 'crazies' in this field and what they mean to the ///. Next month, Al will start a column entitled '/// to the Max'. It will contain information that enables Apple /// users to get the most out of his or her machine!

In this issue I will push out a little more technical information by explaining the format of the SOS Directory Structure and giving examples of what it can be used for. One of these examples is a Pascal Unit and program that allows you to 'Catalog' a disk in Pascal.

In an encore performance from last month, Martin Nichols returns to show you some more things you can do with that second byte

Continued on page 6

Ask THREE: (Letters to the Editor)



Dear Bob:

Congratulations! It's been a long time since a new SIG (Special Interest Group) magazine has come out with a premiere issue as polished as ON THREE. The presentation, content, and general "feel"of the magazine are tops. The graphic information alone would be worth the year's subscription price. All the Apple /// users around here are equally pleased.

There is one thing I would like to comment on regarding color monitors and the ///. I, too, just experienced, first hand, the Horror Story you described with the Amdek Color II. Being a dealer, I was able to do something which apparently you weren't. After seeing the glorious four colors, I sent the thing back to the distributor with a letter to Amdek. We were given credit with no questions asked. Besides the lack of colors, the bandwidth of the Color II is insufficient to present Apple /// 80-column text screen in an easily readable manner. Amdek has a reputation for producing a good dollar vs. quality monitor for the Apple][and the Apple /// computers. The Amdek Video 300 green screen is a good example. Unfortunately, the Color II does a lot to make one forget their better efforts.

There is a fairly simple way to get good color in emulation mode on the Apple ///. Just make up a cable to connect to the Color Video Port, described on pages 132 and 133 of the Owner's Guide. If a DB-15 plug is available, this is the easiest way, but a very workable cable can be made by soldering suitable pins on the shield and center conductor of the cable to the monitor and plugging them into pins 12 and 13 of the plug. Twelve is the NTSC composite video, thirteen is ground. This video output into an Amdek Color I looks just like the same Color I connected to an Apple][. Apple /// video also comes from this connection, but the Color I is hardly readable when looking at the Apple /// 80-column text. (The Color I has considerably narrower bandwidth than even the Color II). Apple /// color, though, e.g., the Dick Cavette show on the ProFile, or the Patterns and Brian's Colors from ON THREE look very good on the Color I.

How do you transfer an Apple][hi-res page to the Apple ///? This is something which is turning up more and more often in demos, so there must be a fairly simple way of doing it. I have heard it's "easy"using Pascal. Just like everything else in the world though, anything is easy if you know how! It would be equally useful to move Apple /// hi-res to the Apple][, making allowances for mode incompatibilities, of course.

Bob, if you maintain the quality exemplified in the first ON THREE, I predict you will have to beat off potential advertisers. While your advertising policy is certainly to be admired, I wonder if ON THREE might not, in the long run, have more to offer subscribers with the considerable amount of money which might become available from an open group of advertisers, rather than from a limited number of carefully screened advertisers. Looking at the matter purely selfishly, it might even be possible to reduce subscription rates and the price of the DOM's.

Issue 2 is anxiously awaited!

Sincerely,

Barry W. Collins
Alabama

Dear Mr. Collins,

Thank you for your letter, it made my day! I hope that in future issues we will be as worthy of your praise.

On the subject of the Amdek color monitor, as I look back I see that I didn't give enough credit to Amdek for their other monitors. Your letter and this line should correct that. However, I still find it unacceptable that Amdek is still advertising their Color II as being compatible with the ///.

A future issue will include a program that allows a two way transfer of Apple][and Apple /// files. Thus, you will be able to move pictures back and forth.

As I said in the first issue, the decision to screen advertisers was difficult. As we get more and more subscribers our prices will drop. Until then, the only thing I can say is hang in there! We will grow - but we will grow at a cautious rate.

If you have any other questions or problems, please don't hesitate in writing again.

Sincerely,

Bob Consorti

Dear Bob,

Thanks for sending me a copy of ON THREE. Not only were we delighted to see a magazine directed (dedicated) to the Apple ///, but we did get some immediate tips from some of the articles.

Please accept the enclosed check to initiate our subscription. We presently have an Apple /// with Monitor ///, Apple Disk ///, UPIC (which will be exchanged for a Grappler×), and an Epson MX-100. We are presently using PFS:File, Advanced Visicalc, and Word Juggler. We'd like to see articles and information on uses in Business and Aviation areas.

Additionally, I have recommended to John Sullivan, the owner of Micro Computer Store in Dayton, that he send a subscription of ON THREE to each purchaser of a /// system as a courtesy gesture.

You've got a winner!

Best regards,

Michael K. Farrell
Ohio

Dear Mr. Farrell,

I'm glad that you could use some information out of the first issue. As we receive articles, we print them, so it may not be very long a wait for the type of articles you desire.

Thank you for telling your dealer about us. We need all the support that we can get, and your idea is great. Even if dealers would just send a postcard out to their Apple /// owners informing them of ON THREE we would grow very fast.

Thanks again for your letter.

Sincerely,

Bob Consorti

Dear Mr. Consorti,

I am overwhelmed. What more can I say. My congratulations on a superior publication. I have not yet read beyond page 4 of my premiere issue, and I am already at the Pascal Editor to write you.

First the comments:

I would like to inform those thinking of buying an Epson MX-80 that a cabling switch has to be made for the UPIC card. A switch must also be installed to allow the 8th data bit to be grounded. This will allow full operation, including screen dumping in both Native and Emulation modes.

Do you have any information on screen dumps for the MX-80. I have heard that the PKASO card will do it, but since I have a UPIC card, I don't want to invest in another piece of hardware. I have heard that a special driver is necessary for using the MX-80 with Business Graphics ///. Do you suppose this driver, plus the appropriate program combination would work?

I would like to change my keyboard layout (something that is touted as being easily done by Apple). Using Visicalc, the "x" could be inserted into the "-" or some other key that doesn't require a shift. I thought of using the Jeppson Disassembler to disassemble the Keyboard Layout file to try to decode it. Do you know an easier way?

Are there any plans that you know of to add Auto Dial, and number storage to ACCESS /// (an excellent package)? I think that this would be a very useful addition to this program.

Sincerely,

Stephan M. Dorman, M.D.
Washington

Dear Mr. Dorman,

Thank you for your letter, again I hope that we will be as worthy of your praise in future issues.

A few other people have responded to the question of the lack of color in the emulation mode. Thank you for your input. The item about the UPIC is notable. It's amazing that Apple named it the 'Universal' parallel interface card. With that name you would expect that you could fix the eighth bit problem from within software.

Anyway, the PKASO card does do screen dumping. One way to do graphics screen dumps is to save the picture on disk and then boot Basic or Pascal and use a program to send the picture data to the printer. We will soon publish programs written in those two languages to do a screen dump to an Epson, thus a special driver or assembly language routine is only needed if you want to print pictures from within Business Graphics.

I've got a few letters out to find out where to get the necessary routines for Business Graphics. When I get a reply I will publish it in the magazine.

I'm pleased to announce that Al Evans, one of our contributing authors, will soon show how to change the Keyboard Layout Table. It will be published in a few months as part of his regular column.

The problem is this - it is not a code file, it is a data file. Because of this, programs like Dr. Jeppsons' disassembler will not work. Without knowledge of the structure of the layout table, the only way to decode it is by trial and error. This obviously takes some time, but we will show you how in just a little while.

I also have a letter out to Apple about updates to ACCESS /// and other Apple /// software. When I know for sure I'll publish it. Don't hold me to it, but I understand that at NCC in June, Apple plans to announce quite a few updates and new packages for the ///.

If I can be of further assitance, please don't hesitate in writing again.

Sincerely,

Bob Consorti

Dear Bob,

I am enthusiastic about the first issue and look foward to more good information in the coming issues. I wish every success to your venture.

How long do you think it might be before something equivalent to Quality Software's "Bag of Tricks" might become available for the ///? That package just saved my hide on a bombed][data disk, so I've learned to appreciate what it can do. Why, you ask, am I emulating when I have a much finer machine available? It's because of a unique $500 digital-image analysis program that has been written only in Integer Basic.

I've got the ProFile hard disk and the Quark series of software, both of which make me very pleased. "Word Juggler" is far better, in my

judgement, that "Apple Writer ///", but everyone has their own preferences. the "Spooler"and "Catalyst" programs from Quark also work very well. To avoid getting a "free" copy of your locked software when loading it onto ProFile with "Catalyst", Quark cleverly "fixes" your master disk so that it will no longer boot — but it can be reloaded onto ProFile via "Catalyst", if that need should arise.

Storing your programs on ProFile does permit you to go from one program to another without rebooting each time. I hope Quark sends you their products so you can do a review (me too! - Ed.). I've found their support to be very good. I had some trouble getting their locked "Word Juggler" to boot on my ///, and they worked with me on the phone to solve the problem.

My retrospective guess is that their method of disk protection makes the booting of "Word Juggler" susceptible to failure on disk drives that run fast. Quark also recommended that I check the drive hub on my /// because some of the earlier versions had hubs slightly out of round. If a hub on the drive is white, they recommended it be replaced with a replacement kit (costs around $12 from Apple dealers) which uses a tan hub. have you heard of any reports of Apple drive problems caused by the white hubs? At any rate, I never did replace mine after discovering that slowing down the slightly fast speed made the booting problem disappear.

Some other new /// programs you might care to review in the future include "Quick & Easy Data Master", "BASIC Extension"and the OMNIPACK (statistics and DBM) series (all are unlocked). These firms deserve our support if for no other reason than leaving their products open for the user to tinker with.

I, too, got stuck with an Amdek Color II monitor thinking it was compatible with the ///. I've got to give Apple some blame for the color-interface problems, however. It is inexcusable to put a signal on the port and label it "RGB color output", when in fact it is not. Does anyone know what Apple had in mind when they designed(?) the color output on the ///? But my story isn't as discouraging—my Color II does "work", after a fashion. It just provides different (and fewer) colors than what the Apple thinks it is generating. How about a review of the relative merits of other RGB monitors on the market—some at rather attractive prices?

My gripe is that after spending $500 dollars for the Microsoft Z-80 CP/M card for the ///, I found that Microsoft's CP/M FORTRAN will not run on it. Oh well, at least I asked first, before also buying the FORTRAN.

A comment on the merits of driving parallel vs. serial printers with the ///: the story I get (though I have not verified it) is that the UPIC output for parallel printers sets bit-8 high, which means that some printers, which use 8-bits (such as the Epson MX-series III Graphtrax-Plus) may have some of their features inhibited. The serial port on the /// can be set up to pass all 8 bits when you use the Systems Utilities Configuration Program. When I first used my Epson printer with the high-bit set, the printer periodically switched back and forth between italics and normal fonts, but the anomalies disappeared when I configured the driver to transmit the 8th bit.

Are there any programs that will run on the /// (including the /// dialect for CP/M) that are directed toward the specific needs of scheduling athletic tournaments consisting of from 6 to 64 teams, including options for rating, ranking, seeding and scheduling the first-round pairings?

How about any programs that are particularly adapted to information management pertaining to small to medium sized fire departments? I'm concerned that I not get involved in reinventing rolling objects.

One last question—is there anything comparable (other than the Pascal utility) for editing Business Basic programs in a manner similar to GPLE for the][? If not, why not?

Sincerely,

John M. Miller
Alaska

Dear Mr. Miller,

Thank you for your letter, I enjoyed it very much.

We are now preparing a 'Stand-Alone' package that will reconstruct blown disks and restore deleted files. Lazarus /// will be available in a couple of months. We have quite a few things planned, so, many such programs will become available in the next few months.

Quark does make very good products and we will be reviewing some of them in the coming months. I haven't heard of any problems caused by the white disk drive hubs. I do believe that the change to tan hubs was economical rather than problem oriented in nature.

Our April-May issue will feature a review of the 'Quick and Easy Data Master' DBMS. Just as you say, we will support programs that are unlocked. This is one of the things that we look for in a review of a piece of software.

Consider the story on the UPIC verified. It does have the problem you describe. You would think with a name like 'UNIVERSAL' parallel interface card, you could set the 8th bit using the System Configuration Program.

There is not a utility presently available that simplifies the editing of Basic programs. The reason is this - Almost before a commercial program is written, the programmer determines how to market it. With Apple /// programmers the problem is not how, but where to market it. Before ON THREE arrived on the scene, people could not mount a successful advertising campaign for their Apple /// products because there was no place they could find Apple /// users. Now they can. In the next few months we will see an explosion in Apple /// software.

I have heard of an Athletic Scheduling program for the ///. I have an inquiry out into where you can get more information. When I receive the answer I will pass it on to you. I don't know of any information management programs that exactly suits your needs, so you will probally have to adapt to a particular software package.

If you have any other questions or problems, please don't hesitate in writing again.

Sincerely,

Bob Consorti

Dear Sirs:

Enclosed is my check for $30. I find your articles and programs excellent. I would like to see more tutorials, such as "Basic - The Easy Way".

I have found a partial solution to Mr. Scattergood's question, (January issue), regarding color in Apple][Emulation. My local Apple dealer, (Byron Johnson, of Candid Computers), rigged up an adapter that combines the RGB signals from the color video port into a composite color signal. This allows me to use an Amdek Color I monitor with my ///. The color quality is good in both Apple /// and in the Emulation mode. The resolution is not high enough for word processing, but this is not a problem, since a color monitor and black-and-white monitor can be used either individually or simultaneously.

Also, I have had an adapter made which combines the two joystick ports into one. This allows me to use a joystick, (Cursor ///), for Apple][games in the Emulation mode.

I have found that some Apple][games will run on the Emulation mode, and some won't. The only way to know if a game will work is to try it out before buying it. Could you have someone test the Apple][games and publish a list of games that are compatible with Apple][Emulation?

Sincerely,

John R. Cade
California

Dear Mr. Cade,

I thank you for your last letter, and I think you'll be pleased to learn that we are planning more tutorials like "Basic - The Easy Way". The Pascal tutorial starts this month, and upcoming will even be tutorials on some of the major application programs!

As an answer to your question, please read the following letter. We received it just a few days after yours. It's amazing how a single forum for information can spread new ideas so quickly.

We're pleased to be of service, and we hope we can help you in the future.

Sincerely,

Bob Consorti

Dear Sirs:

I very much enjoyed the premiere issue of ON THREE and I am looking forward to the next issue. I use my Apple /// in business, microcomputing consulting and sales, and for personal uses and I feel it is the best system available so I was very happy to see your publication.

Recently I acquired a product for my /// that I was quite pleased with, that is the Gameport /// by Micro-Sci, the company that sells disk drives for Apple computers. Gameport /// is a board that plugs into any slot on an Apple /// and enables the user to play any of the Apple][games using an Apple][joystick or hand controllers. I have a TG Products Joystick and their emulation disk but I could only play a few of the games I had. I have not found an Apple][game that could not be played with the Gameport /// installed so I would highly recommend it to everyone. The price for the board and the emulation modifier disk is $74.95.

I read your advertising policy for vendors in your premiere issue and was impressed with the way you will be handling advertising. Thanks again for a great publication, I have already recommended it to several of my friends.

Sincerely,

Roger N. Dietrich
South Dakota

The Editor's Block: continued

of keyboard data. Louis Hanson makes a switch from Visicalc to Pascal, and in this issue starts the Pascal tutorial. Earl Curlson continues his popular column 'Basic - The Easy Way'.

The Apple /// product spotlighted this month is Apple Writer ///. The review may not be as comprehensive as last months', but it will show you many of the program's strong, and (ah!) weak points.

Next time we get the old presses a runnin', reviews will be the big item. Check the New Products Received section for information on upcoming reviews. We will have more tutorials, and who knows what else! Until then, happy ///'ing! ///

The ///'s For Me: continued

what you're doing. I'll answer all mail and trade ideas for ideas, software for software. The Apple /// is a system of unknown potential. Let's make it more than its designers ever dreamed of.

Here's an anecdote which may be apocryphal, but summarizes my point so well I can't pass it up:

A friend of mine was responsible for the video at Steve Wozniak's US festival last summer. Woz stopped in at the trailer to get out of the heat and said something like "Boy, I wish Apples could do graphics like that." My friend had to explain to him that all the graphics for the festival were, in fact, being generated by Apple][']s. ///

The ///'s For Me

by Al Evans

I'm an "old-timer" in this field. I bought my Apple][before disk drives and Applesoft, in the days when the "Red Book" was the only documentation. Back in those dark ages, the only microcomputer programmers were "crazies", people with nothing better to do than disassemble machine language to see what made the monitor tick. The Wizard Woz himself, when he built the original Apple, had no particular intention other than to build a computer which was easy for him to use and which he could program to play Breakout.

The Apple /// is a natural evolution of the][. It's a little faster and has more graphics capacity, along with a great keyboard and an 80-column display. Many of the "tricks" that could be played on the][by means of direct RWTS calls, customized graphics routines, customized routines in the I/O hooks, Applesoft "ampersand" routines, "hidden" routines above the DOS buffers, etc. are "built in" to the /// as SOS calls. In many ways, the /// is a][which has fewer "knobs and switches" at the user/application level but is almost incredibly "adjustable" at the programmer/system level.

However, playing with these knobs and switches is not generally a part of business-oriented systems or application programming. Unless we have "crazies" working with the system to find out what it 'will' do, as compared to what it was intended to do, the capacity of the Apple /// will never be approached, let alone developed as fully as the]['s capabilities have been.

The Apple /// has been out for about two years. Who can tell me how they made those horses run in the original demonstration program? Where's a program to make my computer say anything other that "I'm Okay. Machine status normal"?

What can we do with 256K of memory, anyway? According to my computer, with SOS and Pascal loaded and full graphics, this leaves me with enough free memory space for a complete 64K computer and about 20K left over.

Or, for example, how about this: An Apple /// device driver doesn't necessarily have to drive a device. It's just a machine-language program which can do anything allowed on the "device-driver" level. One thing a device driver is allowed to do is "queue an event"; force the execution of a specified section of code (and "event-handler") after the boot process is completed but before control is handed over to the "system interpreter". An "event-handler" is a piece of machine code which has access to the higher "interpreter" level and is allowed to do just about anything, including make any SOS call.

One possible SOS call sets the .CONSOLE driver to recognize a specified key as an "attention event" so that subsequently, whenever this key is pressed, control will pass to the "event-handler" originally set up by the above device driver.

If this sets off a torrent of possibilities in your mind, you're a "crazy", and the future development of the Apple /// depends directly on you. Be advised that we are few in number and Mom Apple can't support us as she did the Apple][pioneers. Woz is out; she prefers to deal with middle management now. The sad times of cost-effectiveness has beset the computer revolution. Research like this is always hard to justify; you don't know what you'll find or how useful it will be.

Fortunately, none of this changes what your Apple /// is - except to the extent that it affects what you think it is. Nobody knows what microcomputers are for yet! Many pretend to know; they have to in order to be able to say "The microcomputer of 1990 will look like..." or "Sales will increase X% in the business market and Y% in the personal market by 1985..." But let's look at the record:

In the early 1970's, less than ten years ago, competent experts were saying that BASIC could not be implemented on a microcomputer. Two teenagers proved them wrong and became Microsoft. Their BASIC language is now running on nearly every microcomputer made. When I bought my first Apple (can it really be less than five years ago?), the same competent experts were stating categorically that microcomputers were fine for personal use, but just not reliable enough to be used in business.

Even more recently, after I had been using that same Apple in business for almost two years with absolutely no serious problems, capable machine language programmers were telling me that the two games I desired most (a pinball machine and a pool table) could never be written.

Fortunately, they didn't tell Bill Budge or IDSI, because both Raster Blaster and Pool 1.5 became available shortly thereafter. And both of these, along with many other "impossible"programs, are still running on that same Apple][.

So here we are a little later, with the Apple ///. The blind men have decided they definitely know what the elephant is now. They support those who repeat their litany of Visicalc, Word Processing, General Ledger, Accounts Receivable, Accounts Payable, Networking, Business Graphics, data Management... And indeed, computerization of these functions can and will streamline the American way of doing business. But these experts do not and can not recognize the reality that much of the technology which makes it possible, both hardware and software, is the work of "crazies". There is no space on the balance sheet for these "unknown quantities".

Back in 1978, Ted Nelson published and many of us signed a pledge that began: "The purpose of computers is human freedom. I am going to help make people free through computers. I will not help the computer priesthood confuse and bully the public..." Now the public is the user, running pre-packaged software which is mostly written for money by a second-generation computer priesthood, probably including many who signed that pledge four years ago and forgot it. And like the old, many of this new computer priesthood are afraid of those whose objective is to make the computer more useful to the individual by teaching the individual to better use the computer.

So where does that leave us? Relatively down, but not out. We have to fall back and regroup. And mainly, we have to communicate, share the arcana we uncover. For a start, write me and tell me

Continued on page 6

SOS Directory Structure Revealed

by Bob Consorti

In this article I will explain the format of the directory of Apple /// disks. Many exciting avenues will be opened, and I will present examples of what can be done with this information.

If you just want to gain 4 extra blocks of disk space, read the related article 'Disk Pak1' in last month's issue. If you want to see how to list the files on a directory, read the article 'Disk Pak2' in this month's magazine, but if you want to know the why behind the how, read on.

WARNING!!!!!!!!

The reader should feel free to experiment with the concepts shown here, but at all times please keep a back-up copy of any disk you fool with. During the course of the past few months I have ruined many disks with a few simple key-strokes, so I will say again BACK-UP THINE STUFF!!

Format of information on Apple /// Diskettes

To start off, the Apple /// uses the same 5&¼ inch, soft-sectored floppy disks that the Apple][does. As on the][(Dos 3.3), each disk can store 140-K bytes of information. On the /// a small amount of that space is reserved for the directory & booting information, so a total of 136.5-K or 273 blocks are left for the user.

When a disk is formatted (by the disk formatter utility) for use, it is divided into 35 concentric tracks with 16 sectors per track. Each sector can hold up to 256 bytes of info. SOS stores information in two-sector units called blocks, thus each block contains 512 bytes of data. These blocks are numbered from 0-279 decimal or 0-117 hexadecimal.

When the /// is turned on (or whenever you boot a new disk) the system turns on the internal drive and attempts to read block 0 into the RAM at loc. A000 hex. Block 0 contains the second stage boot routine which then takes over and tries to read in the files needed for the operation of the ///.

Thus, on all Apple /// SOS disks, block 0 is reserved for booting information and cannot be used for storage purposes. Likewise, block 1 is considered used but all it contains is two 1's followed by 510 zeroes. In the article 'Disk Pak1' of last month's issue you can learn how to use this and other blocks which normally are reserved.

In addition, each disk contains a directory that tells where files are stored on that disk. On all Apple /// SOS disks, the directory begins on block 2 of the diskette.

The first four bytes in block 2 are very interesting, they are 00 00 03 00. To understand what these bytes mean we must remember that SOS can be asked to 'CREATE pathname,CATALOG,length' where length is the number of bytes you want the directory to be. "Aha" you shout, now you're beginning to see the picture. (If you can see it this quickly you must be on top of things 'cause it took me weeks to get this far.)

On a directory block, the first four bytes are a pointer to the last and next blocks of that directory. I liken it to the Apple][']s directory link-byte. There are 2 bytes per link-info, so if either 2-byte record has zeroes in both bytes there is no link. Any other number indicates the last or next block of the directory (Low-byte, high-byte)

So in block 2, the first 2 bytes (00 00) mean that this is the first block of the directory, and the next 2 bytes (03 00) show that there is more to the directory, and the next block is #0003. The first 4 bytes of block #3 are 02 00 04 00, showing that the last block of the directory was #0002 & the next block of the directory is #0004. Similarly in block #4 we get 03 00 05 00, but in block #5 we have 04 00 00 00. These bytes show the last directory block is #0004 and the third and fourth bytes tell that there aren't any more directory blocks - thus block #0005 is the last block of the main directory.

In the BASIC manual it says that a one block long directory will hold up to 12 files. Since it didn't take into account the directory name (it also takes up a file position) we will say that one directory block can hold 13 file positions. Since we have seen there are 4 blocks in the main directory, simple arithmetic shows that there are (4 blocks X 13 files/block) = 52 file positions available on a standard main directory. 52 - 1 (for the directory name) = 51 which is the number of files the manual says will fit on the main directory. On some of the disks that come with the system this is not the case and only 12 files will fit on a block.

We have just seen that blocks 0-5 are used for booting and directory info. We still need some way of knowing which blocks are used and which are free. Block #0006 contains this information, which I will call the 'block bit-map'. The arrangement of 1's and 0's within each binary byte shows SOS which blocks are used and which are free.

Each byte of block #6 can control the status of 8 blocks of disk space, so 280 blocks / 8 blocks per byte = 35 bytes needed for a normal diskette. If a bit in the block bit-map contains the value 1, the block corresponding to that bit is free. If a bit in the map contains the value 0, the block corresponding to that bit is currently in use. The block bit map for a typical disk might appear as follows:

	1st byte	2nd byte	3rd byte	4th-35th
bits (0-7)	00000001	11101001	11111111	all 11111111 ('FF')
block designated	01234567	89ABCDEF	10 thru 17	18 thru 117

(in hexadecimal)

On this disk we can see that blocks '0-6' are used; '7-A' are free; 'B' is used; 'C' is free; 'D&E' are used; and 'F-117' are all free.

If you are counting along with us, you will see that for a very large disk drive, more than one block is needed for the block bit map. Since one block can be the block map for up to 4096 blocks, a hard disk drive will use a few of them. The largest disk that you will ever attach to your /// will use no more than 8. For these hard disk drives, the block bit map will start in block #0006 and go up to block #000D, depending on how many blocks are needed.

We have already seen how the first 4 (0-3) bytes of block 2 work, so starting in byte #4 are successive 39-byte record entries. I have

discovered 3 different types of file entries: 1- The directory name, 2- A subdirectory name, 3- Regular files.

On the main (or root) directory in block 2, the first file position holds the directory file and it is in the following format. I am going to use the relative byte in my numbering method, that is to say, byte #4 of block 2 (the beginning of the first position in the directory) will become byte #0:

Byte #	Typical Value	Meaning
0	F6	The least significant nibble (LSN) is the length of the file name. See figure 1.0 for an explanation of the MSN.
1-F	42 41 53 49 43 31 00..00 'BASIC1'	The name of the file, or in this case name of the main directory.
10-17	75 00..00	Undetermined
18-19	05 A3	The day, month & year the file was created. See figure 1.1 for an explanation.
1A-1B	00 0C	The minute and hour the file was created. See figure 1.2 for an explanation.
1C	01	The volume number, from 0-255. In this case volume #1.
1D-1E	00 C3	Undetermined
1F	27	The number of bytes per file entry.
20	0C	The number of file entries per block.
21-22	09 00	# of files on this directory. (LB,HB)
23-24	06 00	Location of the block bit-map. (LB,HB)
25-26	18 01	# of blocks on this disk. (LB,HB)

For a subdirectory name entry, the format is the same as the directory name entry except for bytes 23-26:

Byte #	Typical Value	Meaning
23-24	02 00	A pointer to the block number of the start of the directory (or subdirectory) on which this subdirectory is located.
25-26	08 27	Undetermined.

For a regular file entry on any directory the format is as follows:

Byte #	Typical Value	Meaning
0	2A	The LSN is the length of the file name. See figure 1.0 for an explanation of the MSN.
1-F	53 4F 53 2E 4B 45 52 4E 45 4C 00..00 'SOS.KERNEL'	The file name.
10	0C	Type of file. See figure 1.3
11-12	07 00	Starting block of the file. (LB,HB)
13-14	26 00	# of blocks in the file. (LB,HB)
15-17	00 4A 00	# of bytes in the file. (LB,HB, higher-byte)
18-19	75 A1	The day month & year the file was created. See figure 1.1
1A-1B	00 0C	The minute and hour the file was created. See figure 1.2
1C-1D	00 00	Undetermined.
1E	01	File status-locked if this byte < 80, -unlocked if this byte >= 80. **Note:** You can use different values to put a different type of lock on the file.
1F-20	00 00	Undetermined.
21-22	75 A1	Day, month & year file was last modified.
23-24	00 0C	The minute and hour of the files last modification.
25-26	02 00	Block # of the beginning of the directory on which this file exists.

Now how does SOS know where all the blocks of a particular file are on a disk? According to what's been said, the directory information only gives the starting block and the number of blocks in the file.

There are two situations to consider. First, say your file is a short 'HELLO' program of about 250 bytes. As you can see, SOS can store that entire file in one block so no other linking information is needed. But what happens if you have a 5 block long file? The answer is quite simple, bytes # 11&12 of the file's directory entry point to the block containing the block map of that file, which holds the linking information needed if a file is longer than 1 block.

So for our 5 block file, if bytes # 11&12 of the file's directory entry are 4C 00, the block map for that file would be in block #004C. In block #4C we might find 4D 4E 4F 50 51 00..00. In this case the entire file would be in blocks 4D-51. What happens if the block # is > 256? The answer to this question is also very simple. In SOS notation, the block # (0000-FFFF) is represented in the block map by two bytes (LB,HB) where the LB is situated in the 1st 256 byte half of the block & the HB is situated in the 2nd 256 byte half of the block.

Simple arithmetic shows that one block can be the block map of a file with up to 256 blocks in it. That is not enough because SOS is able to handle much larger files. If there are more than 256 blocks in a particular file, byte #'s 11&12 will now point to a block that holds the blocks of the file's block map. Got that!?! Well, for a hypothetical 260 block file whose starting block is #0007, block 7 might contain 08 09 00..00, indicating that blocks 08 & 09 contain the file's block map.

Wait!, you scream. How does SOS know that it's not just a two block long file, but a pointer to the file's real block map? Since I don't know the code structure for SOS.KERNEL I can't tell you exactly, but I think it's a combination of some information in two places. First, the file entry will tell how many blocks is in the file, and from this SOS knows if the file is greater than 256 blocks long the first block is a pointer to the file's block map. Secondly, the MSN of the first byte of the file entry gives the exact information on how to read the file in. Figure 1.0 gives this information on the MSN.

Extra blocks for free (well, almost free)

Since we can't make the disk grow, we are going to have to get those extra blocks I have been promising you. If you remember I mentioned that block #1 contains nothing but two 1's & 510 zeroes. This is the first block we can gain. While marked 'in-use', when I have freed it up and then used it I have had no problems while using the BASIC interpreter, but...

The Pascal system seems to dislike what I have done to my disks. It will store and retrieve info. on the blocks I have freed up but if you list the directory, it will list out all right, but it will then give the message: 'WARNING the directory structure is damaged on this volume:'. Since I have not had any problems besides that warning message in Pascal, I think it is okay to use the increased capacity disks in all applications, and I am doing so.

4 minus 1 leaves 3, so where are we going to get 3 more blocks? You guessed it: the directory! The main directory uses 4 blocks and can hold 52 files (counting the directory name). If we free up all but one block of the directory we will have room for 12 files, which is more than enough for me. With the Apple ///'s hierarchical file structure and ability to create subdirectories, I doubt most people would need more than 12 files on the main directory, but if you want to, you can later add a block or two back to extend your directory.

Wait a minute! What happens if one of the directory blocks you free up has file entries on it? The files will become inaccessible, and will be lost. So to make sure you don't lose any of your files you must do one of the following:

1—If you just want to make data disks that can hold 277 blocks of info., format a blank disk and use the techniques presented later to free up those extra blocks, and then make as many copies of this disk as you will need.
2—If you have a disk that already has programs and or data on it, use the 'Copy files' command of the system utility disk to transfer those files onto a disk that has been 'freed-up'. You can put up to 12 files on the new main directory, but no more!

To free block #1 we must simply change the block bit map to show that it is free. For block #'s 3-5 we must not only change the block bit map, we also have to delete the references to these blocks on the root directory. If we don't do this, SOS will have no way of knowing that those blocks may have been used for storing data, and it could try to store a file entry over important information. There are two ways to do this, first we could write a program, using the Pascal language, to read in the blocks and make the changes needed. While possible (see the article 'Disk Pak1' in last month's issue), it takes a bit of code and even more time. The second way, which we will use here, takes only a minute to update a disk and will provide valuable knowledge of the workings of the ///.

To do this we must learn one more thing about the ///, how to get into and use the monitor mode. When we are in the monitor mode we will have access to the fast 'Read/Write a block' routine which makes this process easy. To get into the monitor mode first SAVE any program you have been working on because we will be resetting the entire system and you will lose any program or data that is in memory. Begin by pressing the 'CTRL' key & the 'Open Apple' key and hold them down while pressing and releasing the 'RESET' button. You should now see a right arrow and a flashing underscore character (the prompt). From here you can enter the 80 column mode by pressing the 'ESC' key, then '8' & finally press 'RETURN'.

We can now read or write a block to a formatted disk in the internal drive. To do this we must understand the command which is:

Blck<Addr1.Addr2Cmd

where 'Blck' is the block # (in hex.) that you want to access,
'Addr1' is the starting address in memory where you want the first block to be read into,
'Addr2' is the end address in memory that you wish to fill,
'Cmd' is the command, either an 'R' or a 'W' for read or write.

For our purposes a similar, but shorter, form of the above command will be used, that is: Blck<Addr1Cmd , where everything is the same as above, but only 1 block can be accessed at once. This will circumvent some of the dangers of this routine because you can only destroy (over-write) one block at a time.

Therefore, the command 0000<2000R will read the 0000'th block of the disk into memory starting at loc. 2000 & the command 0000<2000W will write out the same block we just read in.

Valid block numbers are 0-117 hex., and valid memory locations to read into are from 0C00 to BFFF & from D000 to EFFF. Loc.'s C000-CFFF house the I/O and soft switches like the Apple][, while Loc.'s F000-FFFF contain the monitor and some more soft switches (FFEF-the memory bank select, FFD0-the zero page switch, and

FFDF-the environment byte). Loc.'s 400-BFF are the text screen area, and the first four pages are reserved for zero pages and a stack area.

Getting back to business, format a disk and before putting any files on it place the formatted disk in the internal drive of the ///. Get into the monitor mode and issue the following command: 0002<0C00R this reads block 2 into memory at Loc. 0C00. Byte 0C02 should contain a 03, this is the link-info. that we want to delete, so type 0C02:00 . This change makes the main directory 1 block long but it does not free up the blocks we just 'de-linked'. Since we have finished with this block we can write it back to the disk with the command: 0002<0C00W .

Some people ask me why only the first block of the directory is de-linked, and it is a very good question. The answer is, when you erase the linking info. in the first block you automatically cut off the other blocks in the directory and thus de-link the others also. It's like an Apple /// with 3 external drives, if they are all connected properly everything works. However, if the first extra drive is not plugged in, it and the others connected to it will not be seen by the system and will be considered 'off-line', or just not there.

Now, we only have to modify the block bit map and we will be done. Read it in with the command: 0006<0C00R , and observe loc. 0C00. this is the first byte of the block bit map and, as previously stated, controls the status of the first 8 (0-7) blocks. It can normally have only two values, a 00 or a 01. The 00 tells that all the blocks in that subgroup (0-7) are used while the 01 says that only the 7th block is free. Since we want block #'s 1&3-5 marked free, we must find the hexadecimal equivalent of the binary number 01011100 or 01011101, where the right most bit is a 0 or a 1, depending on the original value.

If the value of byte 0C00 is a 00 we must change it to 01011100 or 5C hex. If it is a 01, change it to 01011101 or 5D hex. Once you have changed byte 0C00 to the new value you can write it back to the disk with the command: 00060C00W . All done!! See, it wasn't too hard.

0006<0C00W

Now, if you boot a BASIC disk and catalog the disk you just updated, you will see that you now have 4 extra blocks of disk space! On an SOS disk only 3 blocks are needed, block 0-the booting info., block 2-the main directory, block 6-the block bit map. A full 138.5K-bytes of storage is now available on each diskette.

There is one more thing that you might have noticed, when you catalogued the new disk, SOS only had to search through 2 blocks of information (the 1 block long directory and the 1 block long block-bit map) instead of the normal 5. This change has resulted in a slight decrease in the time needed to locate a file.

Yes, there is room for one more file!

The most frustrating part of the past few months has been trying to figure out why some disks would only hold 11 files on the main directory. All subdirectory blocks could hold 12, and I was at a loss to explain this difference. Since the first directory block holds the directory name I expected it would hold 12 more files, for a total of 13, but no!!

At the peak of frustration I found that the first block of a subdirectory would hold 12 files in addition to the subdirectory name file. At this point I felt like burning all my SOS disks, and I stopped trying to figure it out for a few weeks. A little while later it came to me and I felt like a real jerk 'cause it isn't that hard to understand.

I compared a subdirectory and a main directory and found that on a main directory, byte 20 of the directory name entry contains a '0C' but on a subdirectory it held a '0D'. Eureka! This byte controls the # of entries per directory block!

Using the methods presented earlier I changed that byte in the directory entry to '0D' and sure enough I was able to store a total of 13 files on the main directory blocks. As a sort of a review, lets go ahead and change it.

Get into the monitor and read in the first block of the directory with the command '0002<0C00R', now enter '0C24:0D' & then write it back to the disk with the command '0002<0C00W'. DONE!!!

Another thing that frustrated me was the fact that the Pascal system for the /// has the same error. Since it had the new SOS 1.1, I expected it to have the correct directory structure, and when it didn't I thought there might be something in there that I couldn't see.

I traced down the error to a single byte on the old SYSTEM.UTILITY disk. One lousy byte had me going without sleep for a few days! Anyway this is only a problem with disks formatted by the old (Version 4.0) of the SYSTEM.UTILITY disk. The new version (which everyone should have by now) does it the right way. However, the new 'BASIC' disk, Version 1.1 also has this problem. I think that the Pascal system disks and the BASIC disks were initially formatted with the old version and were subsequently duplicated, and that is why they have the incorrect structure. You should check byte 24 of block #2 of any disk you may have copied from the master disks. If it's not '0D', change it!

That's all folks. Have fun using the ideas presented here, but remember to try it out on an empty disk first. If you make an error you could easily lose an entire disk. Coming soon I will present a program that does a complete verification of your disks. You will be able to stop those 'Bad Blocks' before they ruin your important data!

Figure 1.0

The MSN of a file's directory entry acts as a flag for SOS in determining how to read the file described in the file's directory entry.

(See the article on the meaning of a file's 'block map' and how SOS reads a file's blocks)

MSN Explanation:

1—Tells that the file's block length is 1 and the block is data.
2—Tells that the file's block length is greater than 1 but less than 256 in length, and that the first block of the file is the file's block map, not data.
3—Tells that the file's block length is greater than 256 blocks in length, and that the first block of the file is a pointer to the files block map.
D—Shows that this file is a 'Catalog' or directory file.
E—Shows that this is a sub-directory.
F—Shows that this is a main directory.

Figure 1.1

Value of Byte # 18, 19	Resulting Date MONTH/DAY/YEAR	Explanation:
00 00	00/00/00	byte #19 → An even LSN sets the months 0-7.
01 00	00/01/00	An odd LSN sets the months 8-12.
01 01	08/01/00	The MSN sets the year according to year = INT (Byte 19/2+.5) & if byte 19 is
00 02	00/00/01	odd year=year-1.
00 A0	00/00/80	
00 A1	08/00/80	byte #18 → This sets the day & month according to month=INT (Byte 18/32) & day=INT
00 A2	00/00/81	((byte 18/32-month) *32+.5) & if byte #19 is odd month = month + 8.
81 A2	04/01/81	

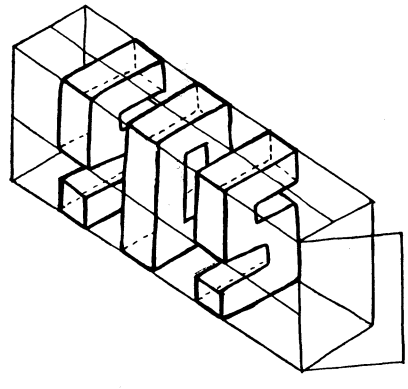
Figure 1.2

Value of Byte # 1A, 1B	Resulting Time HOUR:MINUTE	Explanation:
00 00	00:00	byte #1A → Controls the minute (0-59)
01 00	00:01	byte #1 B → Controls the hour (0-23)
20 00	00:32	
0C 01	01:12	Values greater than 59 or 23 (dec.) are treated as 0 by the Pascal filer, but Basic sees them as
0C 17	23:12	they are, with no checking to find if they are legal.

Figure 1.3

Byte # 10 controls the type of file in the following format:

From BASIC		From BASIC		From Pascal		From Pascal	
Value	Type	Value	Type	Value	Type	Value	Type
00	UNKNOWN	08	FOTO	00	Unknown	08	Fotofile
01	BAD	09	BASIC	01	Badfile	09	Basicprog
02	PASCOD	0A	DATA	02	Codefile	0A	Basicdata
03	PASTXT	0B	WPTEXT	03	Textfile	0B	WPtext
04	TEXT	0C	SYSTEM	04	Asciifile	0C	Sosfile
05	PASDTA	0D	RESERV	05	Datafile	0D	Datafile
06	BINARY	0E	RESERV	06	Datafile	0E	Datafile
07	FONT	0F	CAT	07	Fontfile	0F	Directory



Disk Pak2: Listing Files In Pascal

by Bob Consorti

How many times has this happened to you? Working with a program, you need to retrieve information from a certain file, only you have forgotten its name. No problem right? - all you need to do is catalog the disk. Oh no! You can't do that from within the program. A few minutes later, after you boot a disk that you can catalog a disk with (Basic, Pascal filer...) and see that the darn file is named 'XYZ01.3' instead of 'XYZ01.2' you mutter to yourself - "There's got to be a better way...".

Since these computers are supposed to save us time, there does have to be a better way! - and here it is. A program written in Basic can give the user the option of CATALOGing a directory because the word 'CATALOG' is part of its vocabulary. Pascal, however, has no similar command. True, you can use the filer to list the directory, but we need something that we can use within a program.

The Pascal Intrinsic Unit 'List-Stuff' (Program Listing #1) is the answer. It gives Pascal programs the ability to list the files on any directory or subdirectory. Thus, all programs written in Pascal can now give the user the option of seeing a complete file listing. Extremely versatile - it will send the output wherever you want ('.CONSOLE', '.PRINTER', disk-file...), list the files on an arbitrary sized viewport, and it even returns error messages if anything goes wrong.

'List-Stuff' gives the user the option of pressing the 'ESCAPE' key to exit the listing. Thus, if you are in the middle of a two hundred file listing and want to exit -Press 'ESCAPE'. Another thing that 'List-Stuff' allows is the starting and stopping of the listing whenever a key is pressed. This comes in handy if you want to temporarily stop the listing and don't want to type 'CONTROL-7'.

This Unit is based on the information in the article 'SOS Directory Structure Revealed'. That article is for those of you who want to know the why behind the how, and aren't satisfied with typing in the programs. If you want to learn a little more, by all means read it - you may well get something out of it.

Disk Pak2: Internals

I wrote this routine as an Intrinsic Unit so that all Pascal programs could easily use it. Though, you can easily change it to a Segment Procedure if needed. I tried to make the program as simple and easy to read as possible, so it is not as fast as a Basic CATALOG or a Pascal Filer List command. However, it does give quite a bit of information that those commands do not.

I promised myself not to use assembly language (too hard to read), but promises are made to be broken. In the interests of some kind of execution speed I used the SOS call 'VOLUME' to determine the number of free blocks available on a volume. All of the file information parsing is done in Pascal so you can understand it easier.

Turning now to the Unit 'List-Stuff', we see that the Pascal host program can communicate with the Unit through the means of the PUBLIC procedure 'List-SOS-Directory'. As the listing says, the calling program must give 'List-SOS-Directory' three things. The pathname of the directory you want to list, the pathname of the output file, and the number of lines on the current viewport. The Unit will list the directory (if possible), and return with an error code and message if anything goes wrong.

On an interesting note, I used the compiler option '[$E+]' to allow for provate files within the 'IMPLEMENTATION' of a UNIT. While not mentioned in any of the Pascal manuals supplied with the Pascal language system, the upcoming Apple /// Pascal Technical Reference Manual documents it. By using this option, you will regain the 1K bytes of the I/O buffer used for the file when the UNIT terminates.

The TYPE definitions tell the story well, a Root Entry (the volume name & info.) and a File Entry (the file name & info.) are both records containing the various fields corresponding to the information stored in the disk directory. This is done so that we can read the directory block directly into the record.

While correct on paper, there is a problem. When you PACK a variable in Apple /// Pascal, the fields within the records can not cross a word boundary. Thus, to be packed correctly - all the fields must have a length of a multiple word (an even number of bytes). That's right, each directory record on the disk is 39 bytes long. It complicated matters somewhat, and that is the reason for the MOVELEFT's in the procedures 'Get-root-info' and 'Get-file-info'.

I used a rather complicated routine in opening the output file. While most of the time you will be sending the output to the console, you may want to get a hardcopy of the listing. The procedure 'Set-Out-device' checks if the output file is on a disk and if so it adds a '.TEXT' to the end of it so that it generates a Pascal Textfile and not an Ascii file. Complex I/O error checking assures that the program will not crash, even when you type in a non-existing directory or bad file name.

The code 'CHR(xx DIV 10 + 48) and CHR(xx MOD 10 + 48)' is used extensively throughout the routine to give a formatted screen output. It will write out a two digit number, but instead of leading spaces, it gives leading zeroes.

One interesting piece of information that the routine returns when listing a directory is this: It writes out the number of blocks that the files data occupies and the number of blocks that the entire file takes up on the disk. These are not the same due to SOS using some overhead 'housekeeping' blocks. The routine will also list the number of bytes that the file uses. Since a file may be up to 16 megabytes long, this can be a big number and that is why I used Long Integers in computing them.

Program Listing #2 is the assembly language routine that returns the number of free blocks of the device name passed to it. It is fairly simple and will only return the number of free blocks if a block device name is passed to it. A volume name or subdirectory will not work!

Program Listing #3 is the Pascal program that uses the Intrinsic Unit 'List-Stuff' to list the contents of any directory. This program is also very simple. Notice that we set up a window on the screen with

the dimensions 20X80. We then call the procedure List-SOS-Directory with the appropriate parameters. Notice that the program never properly ends because 2+2 never stops being equal to 4. To leave the program, type 'ESCAPE' and then 'RETURN' when asked for the output file.

To use this Unit, type in Program Listing #1 and compile it with the name 'LIST.STUFF'. Next, type in Program Listing #2 and assemble it with the name 'VOLUME'. Now, use the linker to join together the assembly language routine (Listing #2) and the Unit (Listing #1). Since you may not have that much experience we will go through it step by step.

First, make sure the linker (SYSTEM.LINKER) is available on one of your disks. At the main command level, type the 'L' key - but don't press 'RETURN' yet. When the prompt says 'Host file?', enter 'LIST-.STUFF'. Now if all goes well, the next prompt should say 'Lib file?'. For this one, enter 'VOLUME'. Since these are the only files we want linked, when the prompt again says 'Lib file?', press 'RETURN'. Press 'RETURN' once again when the screen reads 'Map file?'. Finally, when the screen says 'Output file?', type in 'LIST.STUFF'. There, all done! We now have a useable library Unit.

Next, use the librarian to add the Unit to your SYSTEM.LIBRARY. At the main command level X)ecute the file 'LIBRARY.CODE'. When the screen prompts you with 'Output file →', enter 'NEW.LIB'. Make sure that the diskette you are putting 'NEW.LIB' on has enough room to hold the sum of the old library and a little bit more. When the screen prompts you with 'Input file →', enter '.D1/SYSTEM.-LIBRARY'. Now type '=' to copy all the library segments from the old library to the new one.

When the disk drives have stopped making noise, type 'N' then 'LIST.STUFF' and now press 'RETURN'. Now look on the screen under the prompt 'Output file →' and find the first two slot numbers that aren't occupied. Type 1 <space> 1st empty slot number <space> and then 2 <space> 2nd empty slot number <space>. Finally type 'Q' and then enter any copyright notice you want to include. Lastly, use the filer to remove the old 'SYSTEM.-LIBRARY' and to transfer this new library onto your system diskette with the new name 'SYSTEM.LIBRARY'.

Once you have correctly installed it, all of your programs can use it. Finally, type in Program Listing #3 and compile it. You can now execute this file and test the new Unit.

That's about it for now. Next time I will present another handy utility that will allow you to list the files on an Apple][DOS 3.3 diskette from Apple /// Pascal. If you think this is leading somewhere, you're right. In just a little while, you will be able to inexpensively transfer Apple /// SOS and Apple][(e) DOS 3.3 text and graphics files! ///

Disk Pak2: Program Listing #2

```
Return  .EQU    0     ;{ ################################################ }
Volume  .EQU    0C5   ;{ #  Disk Pak2: Get_Volume_Info                  # }
DevName .EQU    0E0   ;{ #  ------------------------                    # }
VolName .EQU    0E2   ;{ #  This assembly language procedure will return # }
TotBlks .EQU    0E4   ;{ #  # of free blocks, the # of total blocks and  # }
FreBlks .EQU    0E6   ;{ #  the volume name of the device name passed to it. # }
ErrPtr  .EQU    0E8   ;{ ################################################ }

        .MACRO  Pop             ;Pull a word from the stack
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM

        .MACRO  Push            ;Push it back on
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM


        .PROC   SOS_Volume,5
        JMP     Begin

SOS_Blk .EQU    *               ;Set up the parameter area
Param0  .BYTE
Param1  .BYTE
Param2  .BYTE
Param3  .BYTE
Param4  .BYTE
Param5  .BYTE
Param6  .BYTE
Param7  .BYTE
Param8  .BYTE

Begin   Pop     Return
        Pop     ErrPtr
        Pop     FreBlks
        Pop     TotBlks
        Pop     VolName
        Pop     DevName
        LDY     #00
        STY     Param4
        STY     Param2
        LDA     #VolName
        STA     Param3
        LDA     #DevName
        STA     Param1
        LDA     #04
        STA     Param0
        BRK                     ;Issue SOS call # C5 (Volume)
        .BYTE   Volume
        .WORD   SOS_Blk         ;Set up pointer to parameters
        STA     @ErrPtr,Y       ;Store the error code high byte
        LDA     Param7
        STA     @FreBlks,Y      ;Store the free blocks high byte
        LDA     Param5
        STA     @TotBlks,Y      ;Store the total blocks high byte
        TYA
        INY
        STA     @ErrPtr,Y       ;Store the error code low byte
        LDA     Param8
        STA     @FreBlks,Y      ;Store the free blocks low byte
        LDA     Param6
        STA     @TotBlks,Y      ;Store the total blocks low byte
        Push    Return
        RTS

        .END
```

Disk Pak2: Program Listing #3

```
PROGRAM List_it;

{ ##########################################################################  }
{ #                                          ---------------------------   # }
{ #  Disk Pak2: List_it                     |  Copyright 1982, 1983 by  |  # }
{ #  -----------------                      |       O N   T H R E E     |  # }
{ #  by Bob Consorti                        |    February-March, 1983   |  # }
{ #                                          ---------------------------   # }
{ #                                                                        # }
{ #  This program uses the Intrinsic unit 'List_Stuff' to list the         # }
{ #  contents of any directory.  Note that you can define the number of    # }
{ #  lines to be listed per page.  Thus, you can set a viewport and list   # }
{ #  the files according to the size of that text window.                  # }
{ #                                                                        # }
{ ##########################################################################  }

USES List_Stuff;            { Contains the routines to list a directory }

CONST Bell = 7;             { Causes a beep on the internal speaker }
      Top_viewport = 2;     { Sets the top of the currently defined viewport }
      normal = 17;          { Sets normal  video output (White on Black) }
      inverse = 18;         { Sets inverse video output (Black on White) }
      Escape = 27;          { The ASCII number of the ESCAPE character }
      Clear_viewport = 28;  { Homes the cursor and clears the viewport }

TYPE Counter = INTEGER;

VAR In_Path, Out_Path: STRING;  { The input and output files }
    Error_msg: STRING;          { Holds the error message and number }
    Error_code: INTEGER;        { returned by the Intrinsic Unit List_Stuff }
    Lines_on_window: INTEGER;   { The number of lines in the current viewport }


PROCEDURE Set_titles;     { Sets the main page heading for the entire program }
VAR i: Counter;

BEGIN
  WRITE (CHR (Clear_viewport));
  WRITE ('Disk Utility Pak2');
  GOTOXY (60, 0);
  WRITELN ('Copyright 1982, 1983');
  WRITE ('by Robert Consorti');
  GOTOXY (64, 2);
  WRITELN ('by ON THREE');
  FOR i := 1 TO 10 DO
    WRITE ('--------');
  WRITE (CHR (Top_viewport))
END; { of PROCEDURE Set_titles }


PROCEDURE Print_Error;  { Routine to print out the error message }
VAR Ch: CHAR;

BEGIN
  WRITELN (CHR (Bell));
  IF (LENGTH (Error_msg) = 0) THEN
    WRITELN ('WARNING: Error #', Error_code)
  ELSE
    WRITELN ('WARNING: ', Error_msg);
  GOTOXY (0,23);
  WRITE (CHR (inverse), 'Press any key to continue', CHR (normal));
  READ (KEYBOARD, Ch)
END; { of PROCEDURE Print_Error }


PROCEDURE Get_Paths;
BEGIN
  WRITELN (CHR (Clear_viewport));
  WRITELN ('(RETURN for ''.CONSOLE'', ESCAPE RETURN to exit)');
  WRITE ('Enter where I should send the listing -> ');
  READLN (Out_Path);
  WRITELN;
  IF (LENGTH (Out_Path) = 0) THEN
    Out_Path := '.CONSOLE';
  IF (Out_Path [1] = CHR (Escape)) THEN
    EXIT (PROGRAM);
  WRITE ('Enter the directory to list -> ');
  READLN (In_Path)
END; { of PROCEDURE Get_Paths }


BEGIN { Main program }
  Set_titles;
  Lines_on_window := 20;   { There are twenty lines on this viewport }
  REPEAT
    Get_Paths;
    Error_code := 0;
    Error_msg := '';
    List_SOS_Directory (In_Path, Out_Path, Lines_on_window, Error_code, Error_msg);
    IF (Error_code <> 0) THEN
      Print_Error
  UNTIL (2 + 2 <> 4)
END. { Of PROGRAM List_It }
```

Disk Pak2: Program Listing #1

```
UNIT List_Stuff;

{$E+ }       { This compiler option allows for private files within the UNIT }

{ ##########################################################################  }
{ #                                          ---------------------------   # }
{ #  Disk Pak2: Directory Lister Unit       |  Copyright 1982, 1983 by  |  # }
{ #  -------------------------------       |       O N   T H R E E     |  # }
{ #  by Bob Consorti                        |    February-March, 1983   |  # }
{ #                                          ---------------------------   # }
{ #  This Intrinsic Unit gives any Pascal program the ability to list the  # }
{ #  files on any directory or subdirectory.  Thus, all programs written   # }
{ #  in Pascal can now give the user the option of seeing a complete file  # }
{ #  listing.                                                              # }
{ #                                                                        # }
{ #  Please read the article and the program List_It to see how to install # }
{ #  and use this Intrinsic Unit.                                          # }
{ #                                                                        # }
{ ##########################################################################  }

INTRINSIC CODE 24; { Only one segment; files are private to this UNIT }
                   { because of the 'E+' compiler option shown above. }
INTERFACE

PROCEDURE List_SOS_Directory (VAR Pathname, Out_Path: STRING;
                              VAR Lines_on_window,
                              Error: INTEGER; VAR Error_message: STRING);

IMPLEMENTATION

{ ##########################################################################  }
{ #                                                                        # }
{ #  The procedure 'List_SOS_Directory' is PUBLIC and can be used by your  # }
{ #  Pascal host program as follows:                                       # }
{ #                                                                        # }
{ #  INPUT to List_SOS_Directory:                                          # }
{ #    1) Pathname - The directory you want to list.                       # }
{ #    2) Out_Path - Where to send the listing.                            # }
{ #    3) Lines_on_window  - The current number of vertical lines in the   # }
{ #                          viewport.  Used to determine where to make a  # }
{ #                          page break when listing to the '.CONSOLE'.    # }
{ #                                                                        # }
{ # Output from List_SOS_Directory:                                       # }
{ #   1) The listed directory.                                             # }
{ #   2) Error - The error code (as indicated by IORESULT) for the last    # }
{ #              completed Input/Output operation.                         # }
{ #   3) Error_message - The STRING consisting of the error message for the # }
{ #                      IORESULT.  If empty, the error is not included in # }
{ #                      in the 'Error_type' ARRAY.                        # }
{ #                                                                        # }
{ ##########################################################################  }
```

```
PROCEDURE SOS_Volume (VAR Dev_Name, Vol_Name, Tot_Blocks, Free_Blocks,
                      Ret_Code); EXTERNAL;

PROCEDURE List_SOS_Directory;
CONST normal = 17;          { Sets normal  video output (White on Black) }
      inverse = 18;         { Sets inverse video output (Black on White) }
      clear_viewport = 28;  { Homes the cursor and clears the viewport   }

TYPE Counter  = INTEGER;                { Following are the TYPE definitions  }
     Byte     = 0..255;                 { that are the building blocks of the }
     Point    = RECORD                  { the directory structure.            }
                  Last_blk: INTEGER;
                  Next_blk: INTEGER
                END;
     Lgth_rec = PACKED RECORD
                  Len: 0..15;
                  Typ: 0..15
                END;
     String15 = STRING [15];
     Filekind = (Unknown, Badfile, Codefile, Textfile, Asciifile, Datafile,
                 Binary, Fontfile, Fotofile, Basicprog, Basicdata, WPtext,
                 Sosfile, F_type13, F_type14, Sos_directory);
     Filler1  = PACKED RECORD
                  w10, w11, w12, w13: Byte;
                  w14, w15, w16, w17: Byte
                END;
     Date_rec = PACKED RECORD
                  Day: 0..31;
                  Month: 0..12;
                  Year: 0..99
                END;
     Time_rec = PACKED RECORD
                  Minute: Byte;
                  Hour: 0..31;
                  Filler: 0..7
                END;
     Vol_num  = Byte;
     Filler2  = PACKED RECORD
                  w1D: Byte;
                  w1E: Byte
                END;
     Entry_info = PACKED RECORD
                    Byte_per_entry: Byte;
                    File_entries_per_block: Byte
                  END;
     Root_Entry= RECORD           { This is the root directory definition }
                   Len: Lgth_rec;
                   Name: String15;
                   Waste1: Filler1;
                   Create_date: Date_rec;
                   Create_time: Time_rec;
                   Vol: Vol_num;
                   Waste2: Filler2;
                   Entry: Entry_info;
                   Files_on_dir: INTEGER;
                   Bit_map_Loc: INTEGER;
                   Blks_on_disk: INTEGER
                 END;
     File_Entry= RECORD           { This is a file definition }
                   Len: Lgth_rec;
                   Name: String15;
                   F_type: Filekind;
                   Start_blk: INTEGER;
                   Num_blks: INTEGER;
                   Num_bytes: PACKED ARRAY [0..2] OF Byte;
                   Create_date: Date_rec;
                   Create_time: Time_rec;
                   Waste1: INTEGER;
                   F_status: Byte;
                   Waste2: INTEGER;
                   Mod_date: Date_rec;
                   Mod_time: Time_rec;
                   Dir_start: INTEGER
                 END;
     Dir_List  = RECORD
                   Last_Next: Point;
                   Root: Root_Entry;
                   Files: PACKED ARRAY [0..12] OF File_Entry
                 END;

VAR Infile: FILE;                           { Directory path to list }
    Device: TEXT;                           { Where to send the listing }
    File_count, Line_count, Count : Counter; { Temporaries used for control }
    Block_buf: PACKED ARRAY [0..511] OF Byte; { Buffer variable }
    File_type: ARRAY [0..15] OF STRING [10]; { An array of file types }
    Error_type: ARRAY [1..127] OF STRING;    { An array of error messages }
    Directory: Dir_List;                    { A directory block }


  PROCEDURE Set_Error_types;    { Sets the error messages }
  VAR i: Counter;

  BEGIN
    FOR i := 1 TO 127 DO                { Null out the strings, }
      Error_type [i] := '';             { or you will get errors. }
    Error_type [2] := 'Bad unit number';
    Error_type [3] := 'Illegal operation';
    Error_type [4] := 'Illegal directory spec';
    Error_type [5] := 'Lost unit - no longer on line';
    Error_type [7] := 'Illegal pathname';
    Error_type [8] := 'No room - insufficient space on diskette';
    Error_type [9] := 'No unit - unit is not on line';
    Error_type [10]:= 'No such file in specified directory';
    Error_type [11]:= 'Duplicate pathname';
    Error_type [12]:= 'Attempt to open an already open file';
    Error_type [16]:= 'Write-protect error - diskette is protected';
    Error_type [36]:= 'Device not available';
    Error_type [37]:= 'Resource not available';
    Error_type [45]:= 'Invalid block number';
    Error_type [64]:= 'Device error - bad address or data on disk';
    Error_type [68]:= 'Can''t find the specified subdirectory';
    Error_type [69]:= 'Volume not found';
    Error_type [70]:= 'File not found';
    Error_type [73]:= 'Directory full';
    Error_type [78]:= 'Illegal access attempted';
    Error_type [80]:= 'File busy';
    Error_type [87]:= 'Duplicate volume error'
  END; { Of PROCEDURE Set_Error_types }


  PROCEDURE Set_File_types;    { Sets the file types }
  BEGIN
    File_type [0] := 'Unknown   ';  File_type [1] := 'BadBlocks ';
    File_type [2] := 'PascalCode';  File_type [3] := 'PascalText';
    File_type [4] := 'Ascii     ';  File_type [5] := 'PascalData';
    File_type [6] := 'Binary    ';  File_type [7] := 'FontData  ';
    File_type [8] := 'Picture   ';  File_type [9] := 'BasicProg ';
    File_type [10]:= 'BasicData ';  File_type [11]:= 'WpText    ';
    File_type [12]:= 'SOSfile   ';  File_type [13]:= 'Datafile  ';
    File_type [14]:= 'Datafile  ';  File_type [15]:= 'Directory '
  END; { Of PROCEDURE Set_File_types }


  PROCEDURE Trap_IO_error;    { If an error occurs, leave the unit }
  BEGIN                       { with an appropriate error message. }
    IF (IORESULT <> 0) THEN
      BEGIN
        Error := IORESULT;
        Error_message := Error_type [Error];
        CLOSE (Infile);
        CLOSE (Device);
        EXIT (List_SOS_Directory)
      END
  END; { Of PROCEDURE Trap_IO_error }


  PROCEDURE Set_Out_device;    { Sets the appropriate output file }
  VAR i: Counter;
```

```
BEGIN
  IF (LENGTH (Out_Path) = 0) THEN
    BEGIN
      Error := 7; { An illegal Pathname }
      Error_message := 'You must supply a pathname!';
      CLOSE (Infile);
      CLOSE (Device);
      EXIT (List_SOS_Directory)
    END
  ELSE
    BEGIN
      FOR i := 1 TO LENGTH (Out_Path) DO
        IF (Out_Path [i] IN ['a'..'z']) THEN
          Out_Path [i] := CHR (ORD (Out_Path [i]) - 32);
      IF ((Out_Path <> '.CONSOLE') AND (Out_Path <> '#1')) THEN
        IF ((Out_Path <> '.PRINTER') AND (Out_Path <> '.SPRINTER') AND
          (Out_Path <> '.PPRINTER') AND (Out_Path <> '#6')) THEN
            IF (POS ('.TEXT', Out_Path) = 0) THEN
              IF (LENGTH (Out_Path) < 11) THEN
                Out_Path := CONCAT (Out_Path, '.TEXT');
{$IOCHECK- }
      REWRITE (Device, Out_Path);
      CLOSE (Device, LOCK);
      Trap_IO_error;
      REWRITE (Device, Out_path);
{$IOCHECK+ }
      Trap_IO_error
    END
END; { Of PROCEDURE Set_Out_Device }


PROCEDURE Open_Directory;    { Open the directory for use }

  PROCEDURE Check_for_valid_directory;    { Determine if it's a directory }
  BEGIN
{$IOCHECK- }
    Count := BLOCKREAD (Infile, Block_buf, 1);
{$IOCHECK+ }
    Trap_IO_error;                    { These are the 'signature bytes'  }
    IF (Block_buf [0] <> 0) OR        { that should always be '0' & '39' }
       (Block_buf [35] <> 39)) THEN   { on an Apple /// SOS directory.   }
        BEGIN
          Error := 7; { Not a valid directory }
          Error_message := 'Not a valid directory';
          CLOSE (Infile);
          CLOSE (Device);
          EXIT (List_SOS_Directory)
        END
  END; { Of PROCEDURE Check_for_valid_directory }

BEGIN { Main of Open_Directory }
{$IOCHECK- }
  RESET (Infile, Pathname);
{$IOCHECK+ }
  Trap_IO_error;
  Check_for_valid_directory
END; { Of PROCEDURE Open_Directory }


PROCEDURE Get_root_info;    { Get the volume information }
VAR temp: INTEGER;

BEGIN
  MOVELEFT (Block_buf [0], Directory, 4);
  WITH Directory.Root DO
    BEGIN
      MOVELEFT (Block_buf [4], Len, 1);
      Name := '      ';
      MOVELEFT (Block_buf [5], Name [1], 27);
      temp := len.len;
      MOVELEFT (temp, Name, 1);
      Vol := 0;
      MOVELEFT (Block_buf [32], Vol, 1);
      MOVELEFT (Block_buf [33], Waste2, 10)
    END
END; { Of PROCEDURE Get_root_info }


PROCEDURE Get_file_info (num: INTEGER);    { Get the file information }
CONST offset = 4;
      bytes_per_entry = 39;
VAR i: Counter;
    low_byte, first, temp: INTEGER;


BEGIN
  first := ORD (num = 12);
  FOR i := first TO 12 DO
    BEGIN
      low_byte := (bytes_per_entry * i) + offset;
      WITH Directory.Files [i] DO
        BEGIN
          MOVELEFT (Block_buf [low_byte + 0], Len, 1);
          Name := '      ';
          MOVELEFT (Block_buf [low_byte + 1], Name [1], 15);
          temp := len.len;
          MOVELEFT (temp, Name, 1);
          F_type := Unknown;
          MOVELEFT (Block_buf [low_byte + 16], F_type, 1);
          MOVELEFT (Block_buf [low_byte + 17], Start_blk, 7);
          MOVELEFT (Block_buf [low_byte + 24], Create_date, 6);
          F_Status := 0;
          MOVELEFT (Block_buf [low_byte + 30], F_Status, 1);
          MOVELEFT (Block_buf [low_byte + 31], Waste2, 8)
        END
    END
END; { Of PROCEDURE Get_file_info }


PROCEDURE Show_root_info;    { Write out the root (volume) information }
BEGIN
  WITH Directory.Root DO
    BEGIN
      WRITE (CHR (clear_viewport));
      WRITE (Device, Name, ' (');
      WITH Create_date DO
        WRITE (Device, CHR (month DIV 10+48), CHR (month MOD 10+48), '/',
               CHR (day DIV 10+48), CHR (day MOD 10+48), '/',
               CHR (year DIV 10+48), CHR (year MOD 10+48), ' ');
      WITH Create_time DO
        WRITE (Device, CHR (hour DIV 10+48), CHR (hour MOD 10+48), ':',
               CHR (minute DIV 10+48), CHR (minute MOD 10+48));
      WRITELN (Device, ') ', 'Volume #', Vol);
      WRITELN (Device)
    END
END; { Of PROCEDURE Show_root_info }


PROCEDURE Show_disk_info; FORWARD;    { Declare these now, so we can use  }
PROCEDURE Do_it; FORWARD;             { them in 'Show_file_info' in case  }
                                      { the user presses 'ESCAPE' to exit }

PROCEDURE New_page (Message: STRING);
VAR Ch: CHAR;
          { Prompts the user to press a key for more files, or to end }
BEGIN
  GOTOXY (0, 23);
  WRITE (CHR (inverse), Message, CHR (normal));
  READ (KEYBOARD, Ch);
  WRITE (CHR (clear_viewport))
END; { Of PROCEDURE New_page }


PROCEDURE Show_file_info (num: INTEGER);    { Write out the files info. }
VAR i, Temp_counter: Counter;
    big_int: INTEGER [8];     { Use LONG INTEGERS in printing out the number }
                              { of bytes (EOF) that the file occupies. }

  PROCEDURE Do_header;   { Write out the title display }
  BEGIN
    WRITE (Device, ' Type       Blks  File Name      ');
    WRITELN (Device, 'Created  Time  Modified Time  Phys  EOF');
    Line_count := Line_count + 5
  END; { Of PROCEDURE Do_header }
```

```
FUNCTION Keypress: BOOLEAN;     { The standard function; tests to see }
VAR CharCount: INTEGER;         { if a key has been pressed.          }

BEGIN
  CharCount := 0;
  UNITSTATUS (1, CharCount, 21);
  Keypress := CharCount <> 0
END; { Of FUNCTION Keypress }


PROCEDURE Check_Keypress;     { Tests to see if a key has been pressed. }
VAR Ch: CHAR;                 { If so, in the case it's an 'ESCAPE' the }
                              { disk info is shown. Otherwise, it waits }
BEGIN                         { until the user presses another key to   }
  IF Keypress THEN            { continue the listing.                   }
    BEGIN
      READ (KEYBOARD, Ch);
      IF (Ch = CHR (27)) THEN
        BEGIN
          Show_disk_info;
          EXIT (Do_it)
        END
      ELSE
        READ (KEYBOARD, Ch)
    END
END; { Of PROCEDURE Check_Kepress }


PROCEDURE Check_console;   { Checks for various options }
BEGIN
  IF ((Out_Path <> '.CONSOLE') AND (Out_Path <> '#1')) THEN
       WRITE ('.')
  ELSE
    IF (Line_count = Lines_on_window) THEN
      BEGIN
        New_page ('Press any key for more');
        Line_count := 2 { The end of a page - so make a new one. }
      END
END; { Of PROCEDURE Check_console }


PROCEDURE PrintLine (i: Counter);     { Writes out one parsed file line }

  PROCEDURE Write_blks_num (num: INTEGER);   { Write out the number of }
  VAR temp: INTEGER;                         { blocks used by the file }
  BEGIN
    temp := num;
    WRITE (Device, CHR (temp DIV 10000+48));  temp := temp MOD 10000;
    WRITE (Device, CHR (temp DIV 1000+48));   temp := temp MOD 1000;
    WRITE (Device, CHR (temp DIV 100+48));    temp := temp MOD 100;
    WRITE (Device, CHR (temp DIV 10+48), CHR (temp MOD 10+48), ' ')
  END; { Of PROCEDURE Write_blks_num }


  PROCEDURE Do_date_recs (Date: Date_rec; Time: Time_rec);

  BEGIN             { Write out one parsed date record in }
    WITH Date DO    { the form Month/Day/Year Hour:Minute }

      WRITE (Device, CHR (month DIV 10+48), CHR (month MOD 10+48), '/',
             CHR (day DIV 10+48), CHR (day MOD 10+48), '/',
             CHR (year DIV 10+48), CHR (year MOD 10+48), ' ');
    WITH Time DO
      WRITE (Device, CHR (hour DIV 10+48), CHR (hour MOD 10+48), ':',
             CHR (minute DIV 10+48), CHR (minute MOD 10+48), ' ')
  END; { Of PROCEDURE Do_date_recs }


  PROCEDURE Do_blocks (F_type: Filekind; Num: INTEGER);

        { Write out the number of blocks that the file itself takes }
        { up.  This does not take into account the header blocks.   }

    BEGIN
      IF (F_type = Sos_directory) THEN
        Write_blks_num (Num)
      ELSE
        IF (Num = 1) THEN
          Write_blks_num (Num)
        ELSE
          IF (Num <= 257) THEN
            Write_blks_num (Num - 1)
          ELSE
            Write_blks_num (Num - ROUND (Num / 257 + 0.5) - 1)
    END; { Of PROCEDURE Do_blocks }


  BEGIN { Main of PrintLine }
    WITH Directory.Files [i] DO
      BEGIN
        IF (F_status < 128) THEN     { The file is locked }
          WRITE (Device, '*')
        ELSE
          WRITE (Device, ' ');
        WRITE (Device, File_type [ORD (F_type)], ' ');
        Do_blocks (F_type, Num_blks);
        WRITE (Device, Name, ': (16 - Len.Len));
        Do_date_recs (Create_date, Create_time);
        Do_date_recs (Mod_date, Mod_time);
        Write_blks_num (Num_blks);
        WRITELN (Device, Num_bytes [0] + Num_bytes [1] * big_int +
                 Num_bytes [2] * big_int * big_int)
      END
  END; { Of PROCEDURE PrintLine }


BEGIN { MAIN of Show_file_info }
  big_int:= 256;
  i := ORD (num = 12);
  IF (i = 1) THEN       { If we get here, it's the first (root) block }
    Do_header;
  REPEAT                          { Until the block is empty or there }
    WITH Directory.Files [i] DO   { are no more files to list. }
      IF (Len.Typ <> 0) THEN
        BEGIN                     { If the file slot isn't empty, print it. }
          PrintLine (i);
          File_count := File_count + 1;
          Line_count := Line_count + 1;
          Check_console;  { Handles end of page and output not to '.CONSOLE' }
          Check_Keypress  { Check to see if the user wants to      }
        END;              { temporarily stop the listing or leave. }
    i := i + 1
  UNTIL ((i = 13) OR (File_count = Directory.Root.Files_on_dir))
END; { Of PROCEDURE Show_file_info }


PROCEDURE Show_disk_info;     { Show the volume information }
VAR V_Name: STRING;
    Tot_blks, Fre_blks, Err: INTEGER;

BEGIN
  WRITELN (Device);
  WITH Directory.Root DO
    BEGIN
      WRITE (Device, File_count, ' Files listed, ');
      WRITELN (Device, Files_on_dir, ' Files on this directory');
      IF (Len.Typ = 15) THEN    { If it's a root volume, get    }
        BEGIN                   { the root volume information. }
          SOS_Volume (Pathname, V_Name, Tot_blks, Fre_blks, Err);
          IF (Err = 0) THEN
            BEGIN
              WRITE (Device, 'Free Blocks: ', Fre_Blks);
              WRITE (Device, '  Blocks Used: ', (Tot_blks - Fre_Blks));
              WRITELN (Device, '  Total Blocks: ', Tot_blks)
            END
          ELSE
            WRITELN (Device, Blks_on_disk, ' Total blocks on this disk')
        END
    END;
```

Program listing continued on page 39

Assembling (ON) the ///

by Martin Nichols

Last month I presented some assembly language functions that made it easier for your Business Basic and Pascal programs to tell if the ENTER key or a key on the numeric keypad had been pressed. One of the functions also returned the second byte of keyboard data. Included was a Basic program that enabled the user to tell the status of all the bits of the second byte.

That program was not too easy to understand, so I am going to implement what that program did as a new group of assembly language routines. Just like last month, these routines can be INVOKE'd from Basic and L)inked from Pascal.

Simply put, we are going to make assembly language functions that will tell the user if certain keys (Closed Apple, Open Apple, Alpha Lock, Control and Shift) are being pressed. Program listings #1 and #2 show how easy these new functions are to use from both Basic and Pascal.

To use these functions, very simple Basic statements like 'Pressed = EXFN%. AlphaLock' will return a one in the variable 'Pressed' if the Alpha Lock key is being pressed and a zero if it's not. Program listing #4 is the documentation for these assembly language functions. It gives a fair amount of information on how to use these routines.

Program listing #3 are the assembly language routines that make these tricks very easy. As you can see, there are eight functions in this module. One for each of the bits of the second byte of keyboard data. Each one is very simple. First they load the accumulator with the appropriate mask for the bit to be tested, and then use the 'BIT' command to compare the mask value with the second byte of keyboard data.

I've received some mail saying that you can already do some of these functions from Basic and Pascal with no assembly language routines, other than the ones that are supplied with the system. This is true, but they are much harder to read and understand.

Part of what the Apple /// does so well is to allow the user to expand the languages of the machine with easily invokable or linkable modules. I think that people would much rather use the one line statement 'Pressed = EXFN%.AlphaLock' over a twenty or so line subroutine to test if the Alpha Lock key is being pressed.

To use these routines in your programs, use the Pascal editor to enter the assembly language routines in listing #3. Assemble it and name it 'KBDFLAG'. The file that the assembler creates will contain the useable assembly language functions, and its name will be 'KBDFLAG.CODE'. To show that this can be used as an invokable module, use the filer to change its name to 'KBDFLAG.INV'.

If you want to test these routines using Pascal, type in and compile program listing #2. Then, use the linker to link them into the Pascal host program. From Basic, just boot a Basic disk and type in program listing #1. Make sure the assembly language routines are available and run it.

Once you have the program running (Basic or Pascal), press any of the special keys and the program will tell you what you've pressed! If you have any questions you can refer to the documentation program (Listing #4).

Next time I will show you how to 'lock up' and 'un-lock' the RESET key in addition to a way to have your /// reboot without having to press 'CONTROL RESET'. Also I will include a procedure that allows you to cut your Apple ///s' speed in half. This last one is for those of you who feel that life is going by too fast.

Until then, write and tell me what you want your /// to be able to do. I will try my best to accomodate you. ///

Assembling (ON) the ///: Program Listing #1

```
0   REM **********************************************************************
1   REM *                                            ----------------------  *
2   REM * Key-Things:  KbdFlag invokable module | Copyright 1982, 1983  |  *
3   REM * (Part 2)     BASIC test program.      |    O N   T H R E E    |  *
4   REM * ------------------------------------- |  February-March, 1983 |  *
5   REM * by Martin Nichols                          ----------------------  *
6   REM *                                                                     *
7   REM * This program will test the state of all the bits of the second     *
8   REM * byte of keyboard data.  Here's the easy way to tell if the Alpha   *
9   REM * lock, control, shift, open apple, closed apple and more are being  *
10   REM * pressed.  Through very simple statements like 'key%=EXFN%.Shift'  *
11   REM * you can determine a wealth of information that your programs can  *
12   REM * use to make life a little easier for the user.                    *
13   REM *                                                                    *
14   REM *********************************************************************
15   INVOKE"KBDFLAG.INV":GOTO 50
20   PRINT CHR$(17+Switch);:PRINT Key.type$(Num):Num=Num+1:RETURN
50   TEXT:HOME:ON KBD GOTO 200
60   VPOS=10:PRINT USING"56C";"<-- PRESS CONTROL-E TO EXIT -->"
70   Key.type$(1)="Special key (Keypad, Arrows, Space, Escape and Tab keys)"
     :Key.type$(2)="Keyboard is on":Key.type$(3)="Closed Apple"
80   Key.type$(4)="Open Apple":Key.type$(5)="Alpha lock":Key.type$(6)=
     "Control":Key.type$(7)="Shift":Key.type$(8)="Any key"
90   Num=1:VPOS=1
100   Switch= EXFN%.Special:GOSUB 20
110   Switch= EXFN%.KybdOn:GOSUB 20
120   Switch= EXFN%.ClosedApple:GOSUB 20
130   Switch= EXFN%.OpenApple:GOSUB 20
140   Switch= EXFN%.AlphaLock:GOSUB 20
150   Switch= EXFN%.Control:GOSUB 20
160   Switch= EXFN%.Shift:GOSUB 20
170   Switch= EXFN%.Anykey:GOSUB 20
180   GOTO 90
200   IF KBD=5 THEN NORMAL:VPOS=12:END
210   ON KBD GOTO 200
220   RETURN
```

Assembling (ON) the ///: Program Listing #2

```
PROGRAM Pascal_kbdflag_test;

{ ##########################################################################
{ #                                          ------------------------    #
{ #  Key-Things (Part2): Pascal test program  | Copyright 1982, 1983 by |  #
{ #  -------------------------------------    |     O N   T H R E E    |  #
{ #  by Martin Nichols                        | February-March, 1983   |  #
{ #                                           ------------------------    #
{ #  This program will check the status of the keyboard special keys and    #
{ #  more using the external assembly language routines that are declared   #
{ #  below.  After compiling this program, use the L)inker to link together #
{ #  the assembly language routines to this host program.                   #
{ #                                                                         #
{ #  When you run this program, see how easy it is to determine the state   #
{ #  of all the modifier keys by pressing them.                             #
{ #                                                                         #
{ ########################################################################## }

CONST Ctrl_E = 5;          { Provide a means for the user to exit the program. }

VAR Num: INTEGER;
    Key_type: ARRAY [1..8] OF STRING;
    Ch: CHAR;


FUNCTION Special: INTEGER; EXTERNAL;      { These are the routines that you  }
FUNCTION Kybdon: INTEGER; EXTERNAL;       { can use to very easily determine }
FUNCTION Closedapple: INTEGER; EXTERNAL; { the state of all the bits of the }
FUNCTION Openapple: INTEGER; EXTERNAL;    { second byte of keyboard data.    }
FUNCTION Alphalock: INTEGER; EXTERNAL;
FUNCTION Control: INTEGER; EXTERNAL;
FUNCTION Shift: INTEGER; EXTERNAL;
FUNCTION Anykey: INTEGER; EXTERNAL;


FUNCTION Keypress: BOOLEAN;        { Tests to see if a key has been pressed. }
VAR Charcount: INTEGER;

BEGIN
  Charcount := 0;
  UNITSTATUS (1, Charcount, 21);
  Keypress := Charcount <> 0
END; { Of FUNCTION Keypress }


PROCEDURE Initialize;                          { Set up the program variables. }
BEGIN
  Key_type [1] := 'Special key (Keypad, Arrows, Space, Escape and Tab keys)';
  Key_type [2] := 'Keyboard is on';
  Key_type [3] := 'Closed Apple';
  Key_type [4] := 'Open Apple';
  Key_type [5] := 'Alpha lock';
  Key_type [6] := 'Control';
  Key_type [7] := 'Shift';
  Key_type [8] := 'Any key';
  Num := 1;
  WRITE (CHR (28))                                  { Clear the screen for use. }
END; { Of PROCEDURE Initialize }


PROCEDURE Display (Switch: INTEGER);     { Show the line in either inverse   }
BEGIN                                    { or normal, depending on the state }
  WRITE (CHR (17 + Switch));             { of the variable 'Switch'.         }
  WRITELN (Key_type [Num]);
  Num := Num + 1
END; { Of PROCEDURE Display }


PROCEDURE Test;     { Perform the test of the second byte of keyboard data. }
BEGIN
  Num := 1;
  Display (Special);
  Display (Kybdon);
  Display (Closedapple);
  Display (Openapple);
  Display (Alphalock);
  Display (Control);
  Display (Shift);
  Display (Anykey);
  GOTOXY (0, 0)                  { Go back to the top of the screen for more. }
END; { Of PROCEDURE Test }


PROCEDURE Exit_Prompt;          { Tell the user how to exit the program. }
BEGIN
  GOTOXY (13, 9);
  WRITELN ('<-- PRESS CONTROL-E TO EXIT -->');
  GOTOXY (0, 0)
END; { Of PROCEDURE Exit_Prompt }


BEGIN { Main Program }
  Initialize;
  Exit_Prompt;
  REPEAT
    IF Keypress THEN
      READ (KEYBOARD, Ch);
    Test
  UNTIL (ORD (Ch) = Ctrl_E)
END. { Of PROGRAM Pascal_kbdflag_test }
```

Assembling (ON) the ///: Program Listing #3

```
; ********************************************************************************
; *                                                    ------------------------  *
; *  Key-Things (Part2): KbdFlag assembly        ¦ Copyright 1982, 1983 by ¦ *
; *                      language routines.      ¦     O N   T H R E E     ¦ *
; *  ----------------------------------        ¦  February-March, 1983   ¦ *
; *  by Martin Nichols                          ------------------------  *
; *                                                                        *
; *  These assembly language routines will enable your Basic or Pascal     *
; *  programs to easily determine the status of all the bits of the second *
; *  byte of keyboard data.  You will be able to tell when the user is     *
; *  pressing the shift key, when they are pressing any of the other modifier *
; *  keys and much more.                                                   *
; *                                                                        *
; *  To use in you Basic programs, assemble this routine using the Pascal  *
; *  assembler, and then invoke it as you would any other invokable module. *
; *                                                                        *
; *  For use in Pascal, declare each of these routines EXTERNAL FUNCTIONS  *
; *  and use the linker to link them into your Pascal host program.        *
; *                                                                        *
; ********************************************************************************

        .MACRO  Pop             ;Pull a word from the stack
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM

        .MACRO  Push            ;Push it back on
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM

        .MACRO  Open
        Pop     Return
        PLA                     ;Discard 4 bytes stack bias
        PLA                     ;(for .FUNC only)
        PLA
        PLA
        LDA     #00             ;Zero the address 'Result' because
        STA     Result          ;you never know what it may contain.
        STA     Result+1
        PHP                     ;save status, then disable interrupts
        SEI
        LDA     Envrmt          ;save environment
        STA     Env
        LDA     #73             ;Use a new environment register
        STA     Envrmt          ;Do this to get at the 'C000' I/O space.
        .ENDM

        .MACRO  Close
        LDA     Env             ;restore environment
        STA     Envrmt
        PLP                     ;restore status (including interrupts)
        Push    Result          ;Give an answer
        Push    Return          ;Come back from non SOS-land
        .ENDM

Return  .EQU    0
Result  .EQU    2
Env     .EQU    4
Kbdflag .EQU    0C008
Envrmt  .EQU    0FFDF

;                       - Function Special -
;
;If a special key (See pages 135 and 165 of the Standard Device Drivers Manual)
;which consists of Escape, Tab, Space, the cursor control keys, and keys on the
;numeric keypad, but not the 'RETURN'!!! key, is pressed, the BASIC statement
;'int% = EXFN%.Special' will return a value of 1 in the variable 'int%'.  If the
;key just pressed was not a so-called special key, the function will return a 0
;in the variable 'int%'.

        .FUNC   Special,0

        Open

        LDA     #80             ;Begin test to see if a special key was pressed
        BIT     Kbdflag
        BEQ     Done

        LDA     #01             ;If we get here it's a special key
        STA     Result          ;Store the character

Done    Close                   ;Time to go home
        RTS

;                       - Function KybdOn -
;
;  If the Keyboard is on (See page 165 of the Standard Device Drivers Manual)
;  the BASIC statement 'int% = EXFN%.KybdOn' will return a value of 1 in the
;  variable 'int%'.  If the keyboard is off, the function will return a 0 in
;  the variable 'int%'.

        .FUNC   KybdOn,0

        Open

        LDA     #40             ;Begin test to see if the keyboard is on
        BIT     Kbdflag
        BEQ     Done

        LDA     #01             ;If we get here the keyboard is on
        STA     Result          ;Save the answer

Done    Close                   ;Back we go!
        RTS
```

```
;                - Function ClosedApple -
;
; If the Closed Apple key (See page 165 of the Standard Device Drivers Manual)
; is being pressed down in conjunction with another key, the BASIC statement
; 'int% = EXFN%.ClosedApple' will return a value of 1 in the variable 'int%'.
; If it is not being pressed, the function will return a 0.

       .FUNC   ClosedApple,0

       Open

       LDA     #20             ;Begin test to see if the Closed Apple key is
       BIT     Kbdflag         ;being pressed.
       BNE     Done

       LDA     #01             ;If we get here the test is true
       STA     Result          ;Save the answer

Done   Close                   ;Back we go!
       RTS

;                - Function OpenApple -
;
; If the Open Apple key (See page 165 of the Standard Device Drivers Manual)
; is being pressed down, the BASIC statement 'int% = EXFN%.OpenApple' will
; return a value of 1 in the variable 'int%'.  If the Open Apple key is not
; being pressed, the function will return a 0 in the variable 'int%'.

       .FUNC   OpenApple,0

       Open

       LDA     #10             ;Begin test to see if the Open Apple key is
       BIT     Kbdflag         ;begin pressed.
       BNE     Done

       LDA     #01             ;If we get here the Open Apple key is pressed
       STA     Result          ;Save the answer

Done   Close                   ;Back we go!
       RTS

;                - Function AlphaLock -
;
; If the Alpha Lock key (See page 165 of the Standard Device Drivers Manual)
; is being pressed down, the BASIC statement 'int% = EXFN%.AlphaLock' will
; return a value of 1 in the variable 'int%'.  If it is not being pressed,
; the function will return a 0.

       .FUNC   AlphaLock,0

       Open

       LDA     #08             ;Begin test to see if the Alpha Lock key is
       BIT     Kbdflag         ;being pressed.
       BNE     Done

       LDA     #01             ;If we get here the Alpha Lock key is pressed
       STA     Result          ;Save the answer

Done   Close                   ;Back we go!
       RTS

;                - Function Control -
;
; If the Control key (See page 165 of the Standard Device Drivers Manual)
; is being pressed down, the BASIC statement 'int% = EXFN%.Control' will
; return a value of 1 in the variable 'int%'.  If it is not being pressed,
; the function will return a 0.

       .FUNC   Control,0

       Open

       LDA     #04             ;Begin test to see if the Control key is
       BIT     Kbdflag         ;being pressed.
       BNE     Done

       LDA     #01             ;If we get here the Control key is pressed
       STA     Result          ;Save the answer

Done   Close                   ;Back we go!
       RTS

;                - Function Shift -
;
; If the Shift key (See page 165 of the Standard Device Drivers Manual)
; is being pressed down in conjunction with another key, the BASIC statement
; 'int% = EXFN%.Shift' will return a value of 1 in the variable 'int%'.
; If it is not being pressed, the function will return a 0.

       .FUNC   Shift,0

       Open

       LDA     #02             ;Begin test to see if the Shift key is
       BIT     Kbdflag         ;being pressed.
       BEQ     Done

       LDA     #01             ;If we get here the Shift key is being pressed
       STA     Result          ;Save the answer

Done   Close                   ;Back we go!
       RTS

;                - Function Anykey -
;
; If Any key (See page 165 of the Standard Device Drivers Manual)
; is being pressed down, the BASIC statement 'int% = EXFN%.Anykey' will
; return a value of 1 in the variable 'int%'.  If it is not being pressed,
; the function will return a 0.

       .FUNC   Anykey,0
```

```
        Open

        LDA     #01             ;Begin test to see if any key is pressed.
        BIT     Kbdflag
        BEQ     Done

        LDA     #01             ;If we get here the test is true
        STA     Result          ;Save the answer

Done    Close                   ;Back we go!
        RTS

        .END                    ;Of Assembly
```

Assembling (ON) the ///: Program Listing #4

```
10  REM ********************************************************************
20  REM *                                                                  *
30  REM *          Kbdflag Invokable Module Documentation                  *
40  REM *                                                                  *
50  REM *           (C) Copyright 1982, 1983 by ON THREE                   *
60  REM *                                                                  *
70  REM ********************************************************************
80  TEXT:title$=CHR$(15)+"----  KBDFLAG INVOKABLE MODULE  ----":GOSUB 300
90  PRINT:PRINT"     Before any Invokable Module can be used, it must be
    loaded into the":PRINT"system by the following Command Format:"
100 PRINT:PRINT")INVOKE KBDFLAG.INV":PRINT:PRINT"where KBDFLAG.INV can be
    the name of this or another Invokable Module.":PRINT:GOSUB 200
110 title$=CHR$(15)+"----  KBDFLAG INVOKABLE MODULE  ----":GOSUB 300
120 PRINT:PRINT TAB(8);"Select documentation on: ":PRINT
130 PRINT TAB(20);"1   Reading the Special key status":PRINT TAB(20);"2
        Reading the state of the keyboard"
140 PRINT TAB(20);"3   Reading the state of the open Apple key":PRINT
    TAB(20);"4   Reading the state of the closed Apple key"
142 PRINT TAB(20);"5   Reading the state of the Alpha Lock key":PRINT
    TAB(20);"6   Reading the state of the Control key"
144 PRINT TAB(20);"7   Reading the state of the Shift key":PRINT TAB(20);
    "8   Reading the Any key status"
146 PRINT TAB(20);"9   End":PRINT
150 PRINT TAB(8);"Which option ";:INPUT a$:x=CONV(LEFT$(a$,2)):
    IF x<0 THEN x=0
160 ON x GOTO 1000,2000,3000,4000,5000,6000,7000,8000,180
170 PRINT TAB(8);"Please enter 1, 2, 3, 4, 5, 6, 7, 8, or 9":VPOS=
    VPOS-2:GOTO 150
180 HOME:END
200 VPOS=24:PRINT USING"76c";"Press any key to Continue.";:GET a$:RETURN
300 PRINT CHR$(14):HOME:PRINT USING"76c";title$:PRINT:RETURN
400 PRINT:PRINT TAB(5)"The Command Format is:":PRINT
410 PRINT")key%=EXFN%.";funcname$:PRINT
420 PRINT TAB(5)"This command should normally be used right after a
      keyboard read such"
430 PRINT"as a GET or INPUT from BASIC or a READ or READLN from Pascal.
      It can also be"
440 PRINT"used from Pascal with the following statement:":PRINT
450 PRINT"FUNCTION ";funcname$;":INTEGER; EXTERNAL;":PRINT
460 PRINT"After being defined in your Pascal program it can be called
      just as any other"
470 PRINT"function. Remember it returns an INTEGER value so you can't
      assign it to a non-"
480 PRINT"integer type variable with out converting it first. From
      Pascal you must also"
490 PRINT"remember to Link before you can eXecute the file.":RETURN
1000  REM --- Special key status
1010  title$="-- Special key --":GOSUB 300:REM Page 1
1020  PRINT TAB(5)"The SPECIAL Function returns a value of 1 if the
        last key pressed was a"
1030  PRINT"special key. A special key is defined on pages 47-49 and
        pages 135-137 in the"
1040  PRINT"Standard Device Drivers Manual. With the exception of the
        RETURN key, whenever"
1050  PRINT"a special key is pressed this function will return a 1. If
        the last key pressed"
1060  PRINT"was not a special key, this function will return a value of 0."
1070  funcname$="Special":GOSUB 400:GOSUB 200:GOSUB 300:REM Page 2
1100  PRINT TAB(5);"You can also use this function from the Immediate
        Execution mode of BASIC"
1110  PRINT"by entering the following statements:":PRINT
1120  PRINT")key%=EXFN%.Special: PRINT key%":PRINT
1130  PRINT TAB(5);"I'll bet you entered the above line by pressing
        RETURN. If you did the"
1140  PRINT"screen will respond with:":PRINT
1150  PRINT")0":PRINT
1160  PRINT TAB(5);"It returned a zero because the last key pressed
        (RETURN) was not one of"
1170  PRINT"the special keys. Now try this! Enter the line again, but
        instead of pressing"
1180  PRINT"RETURN, hit the ENTER key on the numeric keypad. You should
        now be greeted with:"
1190  PRINT")1":PRINT
1200  PRINT"because the last key pressed (ENTER) was on the numeric
        keypad and thus one"
1210  PRINT"of the special keys."
1999  GOSUB 200:GOTO 110:REM Go back to menu
2000  REM --- Keyboard state
2010  title$="-- Keyboard state --":GOSUB 300:REM Page 1
2020  PRINT TAB(5);"The KYBDON Function returns a value of 1 if the
        keyboard is on. I will"
2030  PRINT"admit it's an almost useless function but it is included
        to provide a method of"
2040  PRINT"reading the status of all the bits in the second byte of
        keyboard data. If the"
2050  PRINT"keyboard is ever off (call me if yours ever does it) the
        function will return a"
2060  PRINT"value of 0."
2070  funcname$="KybdOn":GOSUB 400
2999  GOSUB 200:GOTO 110:REM Go back to menu
3000  REM --- Open Apple
3010  title$="-- Open Apple --":GOSUB 300:REM Page 1
3020  PRINT TAB(5);"The OPENAPPLE Function returns a value of 1 if the
        last key pressed was"
```

```
3040 PRINT"the Open Apple key. If the last key pressed was not the
       Open Apple key it will"
3050 PRINT"return a value of 0. This function will only return a value
       of 1 for the time"
3060 PRINT"that the key is being pressed. When the Open Apple key is
       released the function"
3070 PRINT"will return a value of 0.":PRINT
3080 PRINT TAB(5);"The Command Format is:":PRINT
3090 PRINT")key%=EXFNZ.OpenApple":PRINT
3100 PRINT TAB(5);"This command does not have to be used right after a
       keyboard read but it"
3110 PRINT"can be used directly to see if the key has been pressed. It
       can also be used"
3120 PRINT"from Pascal with the following statement.":PRINT
3130 funcname$="OpenApple":GOSUB 450
3999 GOSUB 200:GOTO 110:REM Go back to menu
4000 REM --- Closed Apple
4010 title$="-- Closed Apple --":GOSUB 300:REM Page 1
4020 PRINT TAB(5);"The CLOSEDAPPLE Function returns a value of 1 if
       the last key pressed was"
4030 PRINT"pressed in conjunction with the Closed Apple key. If you
       press the letter 'A',"
4040 PRINT"and at the same time press the Closed Apple key, this
       function will return a 1."
4050 PRINT"If the last key pressed was not pressed at the same time
       as the Closed Apple key";
4060 PRINT"(or after) the function will return a value of 0."
4070 Funcname$="ClosedApple":GOSUB 400:GOSUB 200:GOSUB 300:REM Page 2
4100 PRINT TAB(5);"After this function returns a 1 it will continue
       to return a 1 each time"
4110 PRINT"it is called until another key is pressed that is not
       pressed at the same time"
4120 PRINT"the Closed Apple key is being held. In other words, this
       function will remain on";
4130 PRINT"until it is explicitly turned off.":PRINT
4140 PRINT TAB(5);"Unlike other functions that are 'on' only as long
       as their specific key is"
4150 PRINT"being pressed, this function remains on until another key
       turns it off. The keys";
4160 PRINT"that will turn it off are any of the regular ones that are
       not being held at the";
4170 PRINT"same time as the Closed Apple key."
4999 GOSUB 200:GOTO 110:REM Go back to menu
5000 REM --- Alpha Lock
5010 title$="-- Alpha Lock --":GOSUB 300:REM Page 1
5020 PRINT TAB(5);"The ALPHALOCK Function returns a value of 1 if the
       Alpha Lock key is being"
5030 PRINT"pressed. If the Alpha Lock key is not being pressed the
       function will return a"
5040 PRINT"value of 0.":PRINT
5050 PRINT TAB(5);"The Command Format is:":PRINT
5060 PRINT")key%=EXFNZ.AlphaLock":PRINT
5070 PRINT TAB(5);"This function should be used directly, without
       using a GET or INPUT first."
5080 PRINT"By calling this function you will be able to easily
       determine the state of the"
5090 PRINT"Alpha Lock key. It can also be used from Pascal with
       the following statement:":PRINT
5100 funcname$="AlphaLock":GOSUB 450
5999 GOSUB 200:GOTO 110:REM Go back to menu
6000 REM --- Control key
6010 title$="-- Control key --":GOSUB 300:REM Page 1
6020 PRINT TAB(5);"The CONTROL Function returns a value of 1 if the
       Control key is pressed."
6030 PRINT"If the Control key is not being pressed the function will
       return a value of 0."
6040 PRINT"This function will only return a 1 as long as the key is
       being pressed, as soon"
6050 PRINT"as the key is released this function will return a 0.":PRINT
6060 PRINT TAB(5);"The Command Format is:":PRINT
6070 PRINT")key%=EXFNZ.Control":PRINT
6080 PRINT TAB(5);"This function can be used directly, without using a
       GET or INPUT first."
6090 PRINT"By calling this function you will be able to easily determine
       the state of the"
6100 PRINT"Control key. It can also be used from Pascal with the
       following statement:":PRINT
6110 funcname$="Control":GOSUB 450
6999 GOSUB 200:GOTO 110:REM Go back to menu
7000 REM --- Shift key
7010 title$="-- Shift key --":GOSUB 300:REM Page 1
7020 PRINT TAB(5);"The SHIFT Function returns a value of 1 if the last
       key pressed was pressed"
7030 PRINT"in conjunction with the Shift key. This function will return
       a value of 1 until"
7040 PRINT"another key is pressed that was not pressed at the same time
       as the Shift key."
7050 funcname$="Shift":GOSUB 400
7999 GOSUB 200:GOTO 110:REM Go back to menu
8000 REM --- Any key
8010 title$="-- Any key --":GOSUB 300:REM Page 1
8020 PRINT TAB(5);"The ANYKEY Function will return the value 1 if a key
       was pressed directly"
8030 PRINT"before this function was called. Possible uses include testing
       to see if a key"
8040 PRINT"was pressed during the execution of a program. It's almost
       like the BASIC ON KBD";
8050 PRINT"statement, in that it will tell if a key was pressed during
       program execution."
8060 funcname$="AnyKey":GOSUB 400
8999 GOSUB 200:GOTO 110:REM Go back to menu
```

Products Received

The products outlined below have been received by ON THREE for the purpose of review. Some have been reviewed in the past and many will be reviewed in the future. The products have all been given the ON THREE 'stamp of approval'. This is only an indication that a product works as advertised and is not an endorsement of the product by ON THREE.

PFS: FILE & REPORT

With PFS: File you can create a file, search and update any item or group of items in the file, and print sorted information. Information management at its best, these programs are extremely easy to use.

All PFS products are designed so that a novice can master them in less than an hour. Reviewed in the January issue of ON THREE, these programs received an A- and a B- respectively.

Available from most authorized Apple dealers, these programs are made by Software Publishing Corporation and are priced at $175 and $125 respectively, for the Apple ///.

Software Publishing Corporation, 1901 Landings Drive, Mountain View, California 94043. (415) 962-8910.

QUICK & EASY DATA MASTER

Quick & Easy Data Master is a program that creates custom applications software and report forms designed to your specifications. This package creates an unprotected Business Basic data base program as per your specifications.

The ideal data base program is one that you can design exactly the way you want it: prompts, edits, error messages, headers, titles, computed data, interactive files, report forms, etc. to your specifications. You design it and Quick & Easy will create it for you.

Intended for the more serious computer user who knows how to program in Basic, this package is not very hard to use, but it does require some thought. Sold by Advanced Software Technology, Inc., it is priced at $69.95. To be reviewed in the April-May issue.

Advanced Software Technology, Inc., 7899 Mastin Drive, Overland Park, Kansas 66204. (913) 648-4442.

CRITICAL PATH SCHEDULING

If you are involved in project management and tired of the hassels of project scheduling, the Critical Path Scheduling System is for you! It is a management tool for defining and analyzing the overall concepts of a project and provides a powerful method for scheduling the many tasks necessary to complete the project ON TIME AT THE LOWEST POSSIBLE COST.

Armed with the information that this system provides, the manager is better prepared to make decisions regarding the impact any task will have on the project and permits him to be instrumental in guiding the project rather than just monitoring its progress.

This is a very 'User Friendly' system, and it has an excellent tutorial /user manual. Comprehensive reports make a manager's life a lot easier. Developed by Great Divide Software, it has a suggested retail price of $495. To be reviewed in the April-May issue.

GL-PLUS

To many managers, accounting and the preparation of financial reports are time consuming chores that have to be struggled through. But now, at last, accounting can be simplified.

GL-PLUS is an accounting system designed for the Apple /// computer. It is a flexible, easy to use, journal-based General Ledger system. The computer and GL-PLUS combine to provide you with a tool. A tool to make your accounting chores easier. GL-PLUS automatically guides you through entries and then automatically sorts and posts them.

Report preparation is a "snap" with GL-PLUS. You select the report you wish and the rest is done automatically. GL-PLUS includes a PLUS. The PLUS is a built-in accounts receivable and accounts payable capability that can be implemented anytime you desire.

Another 'User Friendly' system, flexible reporting and ease of use make an excellent accounting package. Developed by Great Divide Software, it has a suggested retail price of $495. To be reviewed in the June issue.

Great Divide Software, Inc., 8060 West Woodard Drive, Lakewood, Colorado 80227. (303) 337-0383. ///

Continued from page 29

```
200   FOR loop.2=10 TO 1 STEP-1
210     VPOS=loop.2:HPOS=15
220     PRINT FN Neg.SQR(value)
230     value= FN Decrement(value)
240     NEXT loop.2
250   VPOS=12:HPOS=24
260   PRINT"- They are the same both ways!"
270   FOR Column=1 TO 80
272     PRINT"-";
274     NEXT Column
276   VPOS=16:HPOS=9
280   PRINT"Some Random numbers"
290   VPOS=14
300   FOR loop.3=12 TO 77 STEP 12
310     GOSUB 60000
320     NEXT loop.3
330   VPOS=16:HPOS=48
350   PRINT"Some more Random numbers"
390   VPOS=14
400   FOR loop.4=12 TO 77 STEP 12
410     GOSUB 61000
420     HPOS=40
430     PRINT temp
440     NEXT loop.4
450   VPOS=21
460   PRINT"What's going on?"
470   PRINT"The second column isn't random!"
480   PRINT:END
60000   PRINT loop.3% FN Rnd.100(-loop.3)
60099   RETURN
61000   temp=loop.4% FN Rnd.100(-loop.4)
61099   RETURN
```

Basic — The Easy Way

by Earl Curlson

To start off, let me apologize for the error in last months' column. On page 25, in the third paragraph of the right hand column, the line '30 LET M% = M% × 1' should read '30 LET A% = A% × 1'. I hope it didn't confuse you too much.

This column is one of a series that is designed to teach you how to use Apple /// Business Basic. Learning a computer language is seldom easy and if you missed the first installment, it will be even harder. Therefore, if you don't have it, I suggest you try to get a copy.

I hope you have been practicing, we have a lot to do today! The program at the end of last months' article is reproduced here as program listing #1. This month we are going to learn all about functions, 'FOR...NEXT' loops and subroutines: So take an hour or so and follow along, you will learn a lot. We will start this month with loops, so please pull up a chair and continue reading!

Looping around

Last month we made a small program that printed out the number from 0 through 99 using an 'IF' statement. You may not have known it, but that was a loop. If you look on pages 110 & 111 of the Basic manual you will see the definition of a loop. Very simply, a loop describes program statements that are carried out over and over again.

The reserved words 'FOR' and 'NEXT' in Business Basic let your programs perform a group of statements a number of times. Pages 111 through 115 show you how to use them. If you look at the first program under the 'FOR and Next' title, you will see it is very similar to last months' small program that counted from 0 to 99. This one counts from 0 to 5.

As it says, there is another way of constructing those types of loops. The next program on that page does just that. Using a 'FOR...NEXT' loop, it also prints out the numbers from 1 to 5. The general form of a 'FOR...NEXT' loop is - 'FOR variable = start TO finish', 'statements to be executed', 'NEXT variable'.

This statement first assigns the number that 'start' represents to the 'variable'. It them performs the statements to be executed and reaches the 'NEXT'. The 'NEXT' statement increments the 'variable' by one (adds one to it) and then sees if it is less than the number that 'finish' represents. If the new value of the 'variable' is still less than 'finish', the statements are executed again. If not, the loop is finished and the program continues with the statements that followed the 'NEXT' (if any).

What all this means is that the group of program statements between the 'FOR' and the 'NEXT' are executed a number of times, determined by the values of 'start' and 'finish'. The control variable must be either an integer or a real number. It can not be a string or a long integer!

The part on the nesting of a loop inside another loop is good. One thing that the manual does not mention is one of the most important things you should talk about when teaching about loops. I hope that this next line becomes engraved inside your head forever. You must never alter the control variable from within the loop.

What the heck does that mean, right? Consider the following program:

```
10 FOR variable = 1 TO 10
20 PRINT variable;" ";
30 variable = variable - 1
40 NEXT variable
```

If you type it in and run it you will get a screenful of the number '1'. Can you see? For this loop to end, the value of the control variable must sooner or later reach the number 10. But because we alter the value of the variable in line #30, it never gets to be 10.

In this small program it is easy to tell why it's not working. However, if you have a program of many hundreds or thousands of lines, you could accidentally change the value of the control variable and this program could give you many headaches trying to figure out why it's not working.

It's not that hard to do either. Just a few months ago I spent two days trying to figure out why a program wasn't working. I was so sure that everything was right! It was driving me crazy until I noticed that I had somehow exchanged two statement lines. That's right! I committed a very bad programming error, I accidentally altered the value of the control variable from within the loop.

If you read through page 115 you will see that a 'FOR...NEXT' loop can get even more fancy. If you've been following along, you may have remembered that even though you can use real numbers for the control variable, when the program hits the 'NEXT' statement, it only increments the control variable by one.

Is there are way to change that? Yes! When used in a 'FOR' statement, the Business Basic clause 'STEP' forces the 'NEXT' statement to increment or decrement the control variable by the amount specified. Pages 114 through 115 show the different possibilities.

You can now use a 'FOR...NEXT' loop to count by 10's or 20's. Backwards, forewards, with fractions - almost anything is possible! To show off your new found skill, write a program that counts from -2 to -5 by .5 and prints out each value. If you have any problems, the answer is just below.

```
10 FOR variable = -2 TO -5 STEP -.5
20 PRINT variable
30 NEXT variable
```

Wow! Would you believe that you just learned all there is to know about 'FOR..NEXT' loops! To become a real expert though, you should test yourself by writing small programs that do nothing more than count. If you take a little time and practice, things will be much easier in the coming months. That's enough for now, take a break and when you come back we will learn about another of Business Basics' powerful features - subroutines!

Subroutines For All

Let's say that you are working on a program and you see that two or more parts of it are exactly the same. In other words, throughout the program, statements are repeated. If you don't want to type them all in in the first place, or if you want to reduce the size of the program, you can use a subroutine.

The Business Basic manuals have a description of subroutines on pages 115 through 118, but it is rather cryptic and not easily understood. If you read the next few paragraphs you should get a good understanding of the 'GOSUB - RETURN' statements.

Much as the 'FOR...NEXT' construct allows you to execute a number of statements over and over, the 'GOSUB - RETURN' mechanism allows different parts of a program to use the same routines. For example, say that throughout your program you need to put a short delay loop. If you had to type it in twenty different times, you would waste precious memory and time.

Using a subroutine, you would only have to type it in once! Whenever you wanted to use it you would type 'GOSUB xxxxx', where xxxxx is the line number of the subroutine. At the end of the subroutine you would put 'RETURN'. Below is an example.

```
10   PRINT "I think I'll waste some time."
20   GOSUB 1000
30   PRINT "Well, that was nice. I think I'll do it again!"
40   GOSUB 1000
50   PRINT "All finished, time to end."
60   END
1000 FOR waste = 1 TO 5000
1010 NEXT waste
1020 RETURN
```

When you run the above program, the computer will print the line 'I think I'll waste some time.' and then seem to stop for a few seconds. Then it will print the line 'Well, that was nice. I think I'll do it again!' and wait a few more seconds before finishing up with the line 'All finished, time to end.'.

What the computer does when it reaches a 'GOSUB' statement is this: It GOes to the SUBroutine indicated by the line number following the 'GOSUB', and it remembers where it came from. When you are finished with the subroutine, the statement 'RETURN' tells the computer to go back to wherever it came from.

You may be wondering by now what line #60 does. As it says, it ENDs the program. If it wasn't there, after the program finished executing the statement at line #50 it would go down to line #1000 and after a few seconds of looping around, it would hit the 'RETURN' statement in line #1020. Since the subroutine wasn't called with a 'GOSUB' and it doesn't know where to 'RETURN' to, it will give an error message.

Just as 'FOR...NEXT' loops can be nested one inside another, subroutines can be nested inside other subroutines. Thus, one subroutine can call another subroutine and so on. Page 116 tells that you can't do this more than 23 times or you will get an error.

Remember that when a 'GOSUB' occurs, the program remembers where it came from. When you have nested subroutines, at times it is necessary to tell the computer to forget where it came from. If this isn't done, the computer may do something you don't want it to do.

The command to make the computer forget this information is 'POP'. It has the affect of jumping out of one level of subroutine nesting. This command is described on pages 117 and 118. Since we aren't going to use it today, I will hold off on a detailed discussion for a little while.

There! We're done with subroutines. For now, that is. We will come back to them in a page or so. We're going to talk about functions in a moment or two, so you may want to take another short break. No, not a week!

Functionally Speaking

Since I am going to write part of the column on them, I guess that I should tell you what a function is! It is just another way to get the computer to give you some type of information. Now, this information will always be in the form of one of the types of information that Apple /// Business Basic understands. A function will return information in the form of either an integer, real number, string or a long integer.

Say that you are an engineer or mathematician and you need to find out the sine of the number 1.9. If you have a calculator you'll probably type in the number and hit the sine key. Looking at what you just did, you gave the calculator a number and you told it to find that numbers' sine.

You supplied a value for the machine to evaluate, we will say that it is called an argument. The machine performed some operation on the value and returned another value.

This is the exact thing that occurrs in Business Basic. A function takes an argument and returns a value. Since it returns a value you must tell the computer what to do with it. For example, the statement 'SIN(1.9)' in Business Basic causes the computer to find the value of the sine of the number 1.9 . But since a function returns a value, where are we going to put it?

That's right - an assignment statement! You can use the line 'A = SIN (1.9)' to store the value that the function returns in the real number variable 'A'. Once you have captured the number you can print it out for the whole world to see with the statement 'PRINT A'.

What's that you're saying? Can I omit the assignment statement and just type 'PRINT SIN (1.9)'? Why don't you try it and see for yourself. Test your computers' limits and see what it can and can't do. That's the only way you are going to learn, so give it a whirl.

By now I think you have a pretty good idea about what functions are, so if you'll turn to page 54 in the Business Basic manual you will see that they have devoted an entire section to functions. That's right - they are very important.

What I want you to do is to start on page 54 and read through page 65. This will give you information and examples on all of the standard functions that the Apple /// uses. Don't worry, I will be right here to help if you have a problem.

Your first problem is on page 55. The line ')WIDTH=*3.3' is a

mistake and should read ')WIDTH=3.3'. See, not even Apple is perfect! Pages 57 through 65 contain all the functions that operate on and return string values. There are quite a few, and with them you can perform most any operation that you would want to do to a string - and more!

Just as a check, you do remember what strings are, right? Good, but for my sanity, I'll repeat it. Apple /// Business Basic can represent what are called 'Strings' by enclosing the text that you want to manipulate within double quote marks. Thus, the statement 'A$= "This is a test string"' tells the computer to find room in memory for the variable with the name 'A' and the type string. Then the computer assigns the words 'This is a test string' to that variable.

Getting back down to earth, in addition to being able to convert from base 10 to base 16 through the use of the 'HEX$' and 'TEN' functions, we have the 'INSTR' function. This very powerful routine will tell you if a string is contained in another string. This is very, very useful for programming. Its' description is on pages 63-64.

While the string functions are very important, we are going to concentrate on another type of function this month. If you look over our sample program you will see a number of functions that work on just numbers - not strings. These are appropriately called numeric functions and are discussed on pages 65 through 70.

The ones we will discuss today are 'SQR', 'INT', 'RND' and the user defined functions. By now you may be thinking that it's an awful lot to remember. You're right, it is. The trick is knowing what to remember and what to look up. Yes, you know you can look up things. Many times I can't remember the exact way to use one of the reserved words. When I can't, I just look it up in the manual.

Getting back to work, the function 'SQR' is described on page 68. It simply returns the positive square root of the number given as an argument. So if you type 'PRINT SQR (4)', the computer will respond with a '2'. Likewise, 'PRINT SQR (5)' will return 2.23607.

As with other functions, you may assign it to a variable. Thus you can use the line 'num = SQR (3)' and 'num' will contain 1.73205. If you assign the value of the square root function to an integer type variable, as in 'num% = SQR (3)', the integer variable 'num%' will contain a 2. As you can see, it rounds the number up to the next highest number. If you try to take the square root of a negative number you will get an error message.

The next function we will discuss is 'INT'. It is described on page 66 of the Business Basic manual. It returns the largest whole number that is less than or equal to the value given as the argument. Thus the statement 'PRINT INT (8.7)' will return an 8, while 'PRINT INT (7.9)' will return a 7. 'PRINT INT (-11.3)' will return a 12. This function is very usefull in removing unwanted numbers after the decimal point.

The function 'RND' is also very useful. It is described on pages 66 & 67. It will return a random positive real number between 0 and 1. For many uses it is very important. Like all of the other functions, you must give it an argument. 'RND' uses the value of the argument in a special way. If you supply 'RND' with a zero, it will return the last random number generated. An example follows.

Type in 'PRINT RND (1)' and you will get some random number between 0 and 1. Say it's .302972. If you type 'PRINT RND (0)' you will get .302972 again. Even though the 'RND' function returns random numbers, using a zero as an argument tells it to return the last random number it made. This argument is called a 'seed' and what the 'RND' function returns is dependent on what you 'seed' the random number generator with.

If the 'seed' is positive, the function will return a new random number each time it is used. If the 'seed' is negative, it will return a random number sequence that will follow the same pattern each time the function is called with a positive argument. Thus, you can force Business Basic to return numbers that are random, but can be repeated if you use the same negative number as the initial 'seed'.

A different sequence is initiated by each new negative argument. The primary reason for this is to initialize a repeatable sequence of random numbers. This is invaluable when debugging a program that uses 'RND'.

Now that we have a few of the basic functions down pat, we are set to tackle the hard one - User Defined Functions! These are functions that the user can create out of any of the built-in functions. This is described on pages 70 through 72.

While hard to describe, they are very easy to make and use. Lets go ahead and make one for practice. How about a function that takes the square root of a number and then adds 1 to it? Why not, right? What we need is something that performs the following function: 'sqr.plus1 (x) = SQR (x) + 1'. Type in the following program:

```
10  DEF FN sqr.plus1 (x) = SQR (x) + 1
20  PRINT FN sqr.plus1 (4)
30  PRINT FN sqr.plus1 (100)
```

If you run the above program, you will get a 3 and an 11 as answers. The program first DEFines a new FuNction with the name 'sqr.plus1' in line #10. Then the function is called first in line #20 with the argument 4. The function then takes over and proceeds to take the square root of the number 4. This result is added with 1 and the result is returned and printed. In line #30, the function is called with the argument 100. Just as before, the function takes the square root of the argument and adds 1 to it. This result is then returned and printed out.

An important thing to learn is that while line #10 defines the new function 'sqr.plus1' with the variable 'x', that variable does not have to be used to communicate with the function. Just as a regular function will work with any value passed to it (not just the number in the variable 'x'), user defined functions allow you to give any variable as an argument.

Say you're an engineer or mathematician and you need to use the hyperbolic functions. At first glance, the built-in functions of Business Basic don't seem to have what you're looking for. However, if you remember that you can build new functions out of the built-in ones, and jog your memory (and some books), you can come up with the necessary formulas.

The hyperbolic sine can be computed with the following function. 'sinh (x) = (EXP (x) - EXP (-x)) / 2'. If you define a function with the line '10 DEF FN sinh (x) = (EXP (x) - EXP (-x)) / 2', you can use the new function with an assignment statement like '20 value = FN sinh

(1.41421)'. After you run the program, the value of the variable 'value' should be 1.93507.

If you look over the examples on pages 70 through 72, you will see that the uses of these user defined functions are almost endless. Practice with these definitions and by all means make up your own! Like I said last month, if you want to learn how to program in Business Basic, you will have to do all the examples I give and more.

Guess what folks? That's right - after all we learned so far, we are ready to look at program listing #1 and understand it! Starting at the top we see that the first four lines are user defined function definitions.

Line #10 contains a function that takes the value of the argument, adds 1 to it and returns the result. Line #20 has a function that does the opposite: it returns the value of the argument minus 1. These are appropriately called Increment & Decrement.

Line #30 is also very simple. It defines the function with the name 'Neg.SQR'. What it does is take the square root of the argument and multiply it by -1. That is all that this function does. I've gotten some mail about negative square roots and their applications, so let me put this to rest. I only intended for the function to take a square root and multiply it by -1, as instructional into what you can do with user defined functions - nothing more.

Anyway, line #40 is a function definition whose function will return a random number from 0 to 99. It uses the argument 'seed' to be the argument for the 'RND' function. This can and does produce some surprising results when used in the program. When the 'RND' function returns a value, it is multiplied by 100 to produce a random real number. I wrote this function to return only whole numbers and that is the reason for the 'INT' statement.

Line #50 clears and moves the cursor up to the left hand corner of the screen. Then the statements in line #60 move the cursor down to the 28th column of the 6th row. Line #70 then prints out some words and we get to line #80. Here the cursor is moved down to the 12th row and line #90 prints out some more words.

Lines #100-140 are the programs first loop. As you can see the control variable (the one that is incremented) is 'loop.1'. This loop goes from 1 all the way up to 10. Line #110 takes the value of the variable 'value', which is initially zero, and uses the 'Increment' function to add one to it. The next line moves the cursor to the first column of the line specified by the variable 'loop.1'.

If you remember, this loop goes from 1 to 10, so what line #120 does is move the cursor from row 1 to 10 on the screen. We do it so that we can print out the numbers at certain places on the screen. Line #130 uses the function 'Neg.SQR' to take the square root of the value of the variable 'value', which will also go from 1 to 10, and multiplies it by -1. After the function is finished, it prints the result in the specified column and row.

Lines #200-240 are almost a mirror (reverse) image of the last five lines. This loop goes from 10 down to 1 (note the STEP value of -1). Line #210 positions the cursor on the line specified by the variable 'loop.2', but just as important, it positions the cursor at the 15th column of that line. This causes the numbers to be printed starting on the bottom row (#10) and working up to the first row side by side with the first numbers that were printed.

Lines #250-260 print out some more text, while lines #270-274 use a 'FOR...NEXT' loop to print out a line of dashes '-' that cuts the screen in half. Lines #276-280 prints out some more words, while line #290 positions the cursor for the next printing.

The exciting stuff starts at line #300. Here we see there is a fairly simple loop whose control variable is 'loop.3'. It starts at 12 and goes up to 77 - but the 'STEP' size is 12. That is to say, as the loop is executed, 'loop.3' takes on the values 12, 24, 36, 48, 60, 72. Line #310 does a 'GOSUB' down to line #60000.

Here we perform one statement and then 'RETURN'. That one statement is fairly complicated though. What it does is print out a number. This number is computed from a multiplication of the result of the function 'RND.100' and the value of 'loop.3'. When it calls the function, it passes as the argument, the negative value of the variable 'loop.3'.

If you remember what the 'RND' function does, when we give it a negative value, it returns the same values over and over when given the same argument. When the loop that starts at line #300 finishes, the loop at line #400 begins.

This loop starts at 12 and goes up to 77, just like the last loop. Line #410 causes the subroutine at line #61000 to be executed. If you look at the difference between line #60000 & 61000, you will see that the only change is in the variable names.

When you run this program, the last two columns that are printed are the same. Since these numbers are random, you would expect that they would be different - but they aren't. If you remember our discussion about the 'RND' function, you will see why these two columns are the same.

That's right! Because they both use the same numbers as arguments, they form a repeating sequence. See, once you know what the words mean, it gets a whole lot easier.

If you don't see how and 'why' everything works, study the program a little more. You have to practice or you will never learn Business Basic. That's all for this time. Next issue we will take a closer look at strings and learn about Apple Business Basic I/O (input/output). If you want to work ahead, read the parts of the manual concerning strings. That should be enough to satisfy you until next time! Coming up soon we're going to make a 'Hello' program that will perform many useful functions. Until them, remember - if you get discouraged, don't give up - try harder! ///

```
10   DEF FN Increment(num)=num+1
20   DEF FN Decrement(num)=num-1
30   DEF FN Neg.SQR(num)=SQR(num)*-1
40   DEF FN Rnd.100(seed)=INT(RND(seed)*100)
50   TEXT:HOME
60   VPOS=6:HPOS=28
70   PRINT"Negative square roots from 1 to 10"
80   VPOS=12
90   PRINT"Going down....Going up"
100   FOR loop.1=1 TO 10
110     value= FN Increment(value)
120     VPOS=loop.1:HPOS=1
130     PRINT FN Neg.SQR(value)
140     NEXT loop.1
```

Continued on page 25

ON Pascal

by Louis Hanson

Hello there! This column is for those of you who want to learn the remarkable Pascal computer language. I will teach, guide and prod you until you have a good working knowledge of what Pascal is - and what it can do for you. Along the way we will learn how to use all the elements of the Apple /// Pascal system, including the Editor, which doubles as a fine word processor.

As a start, we should first think of why we need to use Pascal. Perhaps you ascribe to the belief that Basic causes brain damage, and would like to keep your sanity. Maybe it's because many of the programs contained in this magazine are written in Pascal, and you want to know what you are typing in. Or quite possibly it's that you just want to learn new things and you hear that Pascal is 'THE' computer language to use.

For whatever the reason, I am glad that you are still reading, it must mean that you have some interest. Why Pascal? Good question! I hope I can give a good answer. Before we get down to business, let me say that this column is not just for people who already know how to program. Indeed, it may be easier for those of you who don't have other computer language quirks to unlearn.

The computer language Pascal was defined by computer scientist Niklaus Wirth. His initial objective was to make a language that could be used to teach students good programming skills. As its popularity spread, other people began to see that Pascal was a very good language.

At the University of California at San Diego, a very important version of Pascal was developed. Called UCSD Pascal, it is the basis of Apple /// Pascal. It uses what is called 'P-Code', which will be described in a few lines. One of the wonders of Pascal is that it can be transported from one computer to another very easily. Since UCSD Pascal is available on just about every computer made, most any program you write on your computer can be used on another computer that has UCSD Pascal.

This opens tremendous possiblities for software marketing. Write one applications package and you can use it on scores of computer systems. As some advertisements say, the same pacakge can run on an Apple to a Zenith. While there are some limitations, this does describe the abilities of UCSD Pascal.

Sounds pretty good, right? Well, before you do run out and spend about $200 on the Apple /// Pascal package, let me tell you some more of its good and bad points. Yes - it does have some bad points! The minimum configuration is a 128K Apple /// with at least one external disk drive. Two external drives are recommended, while a Profile hard disk is best.

The Apple /// Pascal system consists of three diskettes that contain all the files needed to implement UCSD Pascal on the ///, and four instruction manuals that tell you just about everything you need to know about the package. One of the nicest features is the Editor. Used to create and modify the program statements you type in, it can be used to write letters and do other word processing tasks.

If you have used Basic, you know that you can type 'PRINT 821 + 180', press the 'RETURN' key and the result '1001' will be immediately printed on the screen. To get the same result in Pascal takes a bit more of work. You need to know how to use the Editor and Compiler to get it to work. The Pascal program to add 821 and 180 is below.

```
PROGRAM Addition_Test;
BEGIN
  WRITELN (821 + 180)
END.
```

To make it work, you have to use the editor to create the lines of text above. Then you have to save the file on disk and exit the editor. Next you must compile the program to a form the computer can execute. Finally, you Run the program to get the answer (1001).

Seems like a lot of work, doesn't it? If you make an error in typing in a Basic statement, it is usually immediately detected and you simply retype the line. With Pascal, the error is not found until you compile the program (or later). If you want to fix it, you must go back to the editor and fix the mistake. After fixing it, you then have to re-compile it.

As you can see, Basic is better for short programs. As programs get longer, in Basic they get harder to read and understand. One of Pascal's big points is its readability. Because Pascal allows (and encourages) many comments, it is much more legible as the size gets bigger than a comparable Basic program.

By now you must be asking yourselves, "What is this Compiling bit about?" The answer is a little complicated and will take a few paragraphs. The Basic language is called an interpreter because when a program is "RUN", the system scans each line of your program for words it understands. As it finds these words, built in instructions are called that performs the action that those words mean. For example, when the Basic interpreter sees the line PRINT "Hello", control is passed to a section of the interpreter that handles 'PRINT' statements. Here, the print instruction is turned into a set of machine language instructions that the computer can execute directly.

This scanning and interpreting takes up a lot of time. Consequently, the time that a Basic program takes is much slower than what a pure machine code version of the same program would take. With Pascal you must first type in the program and then Compile it. The Compiler converts your Pascal statements into a set of instructions that the computer can use directly. These instructions are called the 'object code', while the original statements are called the 'scurce code'. When you Run a Pascal program, it is this 'object code' that is executed. Thus a Pascal program will be able to go faster than its Basic counterpart because there is little scanning and interpreting, just execution!

If you noticed I said that the Compiler produces 'object code' and not machine language. This is due to the fact that we are working on a P-Code version of the language. For all of you who have been waiting for me to explain what P-Code means, here goes!

When UCSD pascal was in its infancy, it was determined that it would take years to write a Compiler for each different computer that produced machine language that each individual computer could execute. It was decided that the UCSD Compiler would put out 'P-Code' instead of machine code. Compilers would put out the same P-Code regardless of the computer used. Thus, only one general Compiler was made for all computers.

This 'P-Code' is a set of instructions that needs to be interpreted (like Business Basic) to work. The P-Code interpreter (SOS.INTERP on the Pascal system disks) analyzes each P-Code instruction and performs the matching action for that instruction. Since the P-Code has already been scanned for errors during compilation, the Pascal interpreter can work much faster than the Basic interpreter.

Each computer can use the same general Pascal Compiler, but each must have its own interpreter. Big deal, huh? We traded writing a Compiler for just our machine into writing an interpreter for just our machine. Well, it seems that the interpreter is very simple to write for any machine. Because each computer system compiles into the same P-Code, compatibility is not a problem. With minor restrictions, you can execute the P-Code of one computer on another.

Since UCSD Pascal was so easy to adapt to all systems, its popularity has spread until today most every micro and mini computer has it. Now it's about time to get started. This month we will learn how to use the Editor and Filer. We will also write our first Pascal program. So kick off your boots, roll up your sleeves and let's get cracking!

The Filer

Once you get the Pascal package for the Apple ///, the first thing you should look at is the 'Introduction, Filer, and Editor' manual. It holds most of the information needed to start using Apple /// Pascal. If you turn to the Preface you will see that this is the book to start with. Reading the overview in chapter 1, you should follow the instructions and make copies of your Pascal system diskettes.

Also in chapter 1 you will find information on rearranging the files on the diskettes. This is very important! If you have a Profile hard disk drive, follow the instructions in the Profile manual to put the Pascal files on the hard disk. This will result in a tremendous speed increase when operating the Pascal system. If you only have one external disk drive (excluding high density drives), you will find working conditions impossible. The only remedy is to buy another disk drive. C'mon, the prices aren't that high!

If you don't have a hard disk drive, to use this article series as a tutorial, you must transfer the file 'SYSTEM.FILER' from PASCAL1 to the disk in your second external drive and then delete 'SYSTEM.FILER' from PASCAL1. Since you can't delete the file by using the Filer (it's sort of like committing suicide), use the System Utilities Disk to transfer and delete the file. The reason for this is we are going to need some space on PASCAL1 to put programs, and with the Filer there, there just isn't enough room.

Since this is your introduction to Apple /// Pascal, we should start at the begining. As with all languages on the ///, you have to boot the appropriate disk to use the language. Insert the copy of PASCAL1 into the built-in drive and either turn on the computer or press 'CONTROL RESET'. In about thirty seconds a screen like the one on page 4 will appear.

On pages 4 and 5 you will also find information on the 'Prompt lines'. The mail level command prompt line is the single most important place in Apple /// Pascal. From here you can invoke and enter the Editor to make your Pascal programs, the Filer to perform file operations, the Compiler to translate the program you wrote with the Editor to a form the computer can execute, the Assembler to make assembly language routines that your Pascal programs can use, and the Linker to link together your Pascal and Assembly language routines.

Chapter 2, "The Command level", describes all of the options available from the command prompt line. It may not be the best of reading but it does have some important information. We aren't going to learn all of those commands today, but you may want to look it over for the future.

Right now we are going to start using the Pascal system, so you should have booted the system as described a couple of paragraphs ago. When the command prompt line appears, press the 'F' key once and see what happens. The disk drive will make some noise and in a few moments the Filer prompt line will appear. Chapter 3 describes the operation of the filer, so open the manual to page 30 and follow along.

As the chapter says, the Filer manipulates files. The first few pages give a brief overview of the options of the filer and their use. The rest of the chapter describes in depth the options and how to use them. In the next few paragraphs we will discuss these options and what they are for.

In many ways the Filer is a 'mini' System Utilities Program. Indeed, it does most of the things that the System Utility Program does, and a few that it doesn't! Just as the System Utilities Filer can list the files on a directory, the Pascal Filer can do the same thing. At the Filer command level, press the 'L' key once. Pages 47 to 52 tell how to use this command. Generally, just type in the pathname of the directory that you want to list. After typing 'L', enter something like '.D1' or '.D2' and the specified directory will be listed to the screen. You can also send the listing to a printer or disk file by entering a destination file specification as described on page 51. If you want to send a listing of the files of the disk in the internal drive to the printer, you would enter (after typing 'L') '.D1, .PRINTER'.

If you have a hard disk storage system, you probably use subdirectories. When listing the contents of a directory with the list command, at times it is helpful to see the contents of all the subdirectories. The 'E' command of the Filer does this. It is described on pages 51 and 52 of the manual. At the Filer command level, type the 'E' key and enter the directory name. Just like the 'L' command, you can send the listing anywhere.

If a device driver with the name '.PRINTER' is not configured into your system, you will get an error message when trying to print to it, as in the above commands. How can you find out what devices are configured into your system? You could use the System Configuration program to do it, but the Pascal Filer has a command that allows you to see all volume names, device names and the device numbers of all the input and output devices whose drivers have been configured into the system. To use this command, from the

Filer command level type 'V' for Volumes. This is described on pages 46 to 47.

The Filer also allows you to transfer files from one place to another. These transfers can be quite complex and they are described on pages 52 through 61. You can copy file to file, volume to volume, subdirectory to subdirectory and more! Since we won't be doing much transfering yet, I'm not going to spend much time discussing this command. If you want to learn everything there is to know, read the above mentioned pages and follow the examples.

You can also use the Filer to create new subdirectories and reserve space on a disk for a file. This command can be used by typing 'M' for make. Page 62 and 63 tell all the options of this command. When you're finished reading those pages, give yourself a test. Try to use the Make command of the Filer to create a subdirectory named 'TEST.DIR'. After typing 'M' you should enter '.D2/TEST.DIR!'. This will create a one block long subdirectory named 'TEST.DIR' on the disk in drive #2.

The Change command allows you to rename any or all of your files. It is described on pages 63 to 68. After reading over that information, use the command to rename the subdirectory you just made with the name 'NEW.SUBDIR'. The command sequence is 'C' - to invoke the Change command, '.D2/TEST.DIR, .D2/NEW.SUBDIR'

The Filer option 'R' for Remove, lets you delete files from your diskettes. After reading over the decription on pages 68 to 70, use the command to delete the subdirectory we have been working with. By now you should be able to figure it out without my saying how, so I won't!

Since we aren't going to use Apple][formatted disks, I am not going to discuss the Krunch or eXamine commands. One of the ones we are going to discuss is the Zero command. It will delete all the files on a directory or subdirectory. The description is on pages 71 and 72. Since this command deletes one file at a time, it takes a while. Therefore, you should limit its use to erasing the files on a subdirectory. This is because it is much faster to just format the disk if you want to erase all the contents.

The Prefix command of the Filer is very useful. It is described on pages 73 through 75. As the manual states, it is a big timesaver. You can set the Prefix to whatever subdirectory you are working on and then only refer to the local pathname. My hard disk has a subdirectory named 'PASCAL' and on that subdirectory is another subdirectory with the name 'WORK.AREA'. This is where I store all the Pascal files that I am currently working on. When I boot up Pascal, I use the Filer to set the Prefix to '.PROFILE/PASCAL/WORK.AREA'. After doing this, I can simply use the local name (the name of the files in the subdirectory 'WORK.AREA') when Editing, Compiling etc. the files I am working on.

The Filer also allows you to read and change the system clock with the Date command. Even if you don't have a working clock installed in your system, the Pascal Filer will remember the date that it was last used, and use that when saving or updating file information. To read or change the date, look over pages 75 and 76 and then press 'D' at the Filer command level. Now enter the current date and press return. You should see the disk in the internal drive come on for a second or two while the date information is stored on disk. Yes, that's how the computer remembers the last time it was used. This date information is stored in the file 'SYSTEM.MISCINFO' on the boot disk.

The Alter command of the Filer can be used to change a files type, write-protect status and date of last modification. It is described on pages 76 and 77. To use it, type 'A' at the Filer command level and then follow the instructions as listed on the above mentioned pages. For now you will probably only need this command to write protect your files so that you can't accidentally delete them.

One handy utility option is the Bad Blocks command. It will find any blocks on your diskettes that are damaged and not useable. Described on pages 84 to 86, it is very similar to the Verify option of the System Utility program. You should use it everytime you format a disk, to check for flaws.

Once you're finished using the Filer, you can use the Quit option to return to the main command level. To use it, just type 'Q' and the main command level prompt line will come up again. In a few paragraphs we will learn about the workfile and its related commands, but for now we're done talking about the Filer.

That's it! You now know how to use just about all of the commands of the Filer. You're right, it is a lot of information. But if you want to learn Pascal, you're going to have to read and read and read. And when you finish reading, you have to test your knowledge by practicing. If I could hold your attention this far, you must have enough interest to practice. Before we continue, first take a break. You deserve it! When we start again, we are going to jump on over to the Editor and learn how to use it to write Pascal programs!

The Editor

Before trying to use the Editor, make sure you have a copy of PASCAL2 in the second disk drive. At the main command level press the 'E' key once. This invokes the file 'SYSTEM.EDITOR' from the disk. Chapter 4 (pages 92 through 154) describe the functions of the Editor. This chapter contains very helpful information for learning how to start using the Editor.

If you read through the first three pages you will come to a page titled 'Starting a New File'. If you have been following along, your screen should look like the one under the first paragraph of page 95. Since we are starting a new file, when you get this prompt, just press 'RETURN'. Congratulations, you're now 'in' the Editor.

Just as the Filer and the main command level have their own prompt lines, the Editor has one. It is shown on page 94. As with the other prompt lines, to use one of the options, simply press the appropriate key. This month we are going to learn the 'I, D, R, A, J and Q' options. These are the Insert, Delete, Replace and Adjust keys, and the Jump and Quit commands.

Press the 'I' key once to get into the insert mode. If you look at the prompt line you will see that it has changed as shown on page 95. These are the options available at this level. You can type in text, use the left arrow key to remove the character to the left of the cursor, press 'CONTROL X' to delete an entire line, 'CONTROL C' to accept the text you just typed in, and 'ESCAPE' to return to the main prompt line of the Editor without accepting the text you just typed.

The best way to learn is to try, so at the Editor command level press

'I' to insert some text. Now enter the lines below, remembering that at the end of a line press 'RETURN'. Don't forget the semicolon at the end of the first line!

```
PROGRAM Addition_Test;
BEGIN
  WRITELN (821 + 180)
END.
```

After you type in the line 'WRITELN (821 + 180)' and press 'RETURN' you will have a problem. The cursor will drop down and be flush with the 'W' in 'WRITELN' instead of being flush with the left side of the page. To make the cursor be flush with the left edge of the screen, type 'CONTROL C' to get to the Editor command level and then 'A'. This is the Adjust command. As the new prompt line says, you can press 'L' to move the cursor to the left of the screen, 'R' to move it to the right, 'C' to center the line the cursor is on, or use the arrow keys to move the line. For our example, press 'L' to move the cursor over to the left of the viewport and then press 'CONTROL C' to exit the Adjust mode.

When you finish the last line ('END.') press 'CONTROL C' to have the computer accept the text you just typed in. As soon as you do that, you will again see that Edit prompt line. This means that the computer has stored the lines you just typed in. Since we would like to see the computer execute the program we must compile it. Type 'Q' for quit and you will be greeted with a screen like the one on page 102. These are the options of the Quit command.

Since our program is small and doesn't take up much space, we can use the workfile to store our programs. Type 'U' for Update. Once you do this, the disk drive will make a little noise and then stop. You just saved the lines you just typed in to the disk file with the name 'SYSTEM.WRK.TEXT' on your boot disk. This is a very special file, which we will discuss further in a moment.

Now that the file is stored on disk, you can exit the Editor with a press of the 'E' key. Again, the disk should make a little noise and the main command level prompt will appear. Since we have to translate the lines we just wrote to a form that the computer can execute, we must Compile the program. To do this, simply press the 'C' key.

The computer will search for the file 'SYSTEM.COMPILER' on all of the disks in the system. If it doesn't find the file, you can't compile your program and you will get an error message. If all goes well, some strange lines will appear telling you that the Compiler is running and it is translating the lines of text you wrote into a form that the computer can execute. If all goes well, in a few moments the main prompt line will appear again. This tells you that you made no typing errors that the Compiler could detect and the program is read to Run.

You may be wondering how the Compiler knew what program to compile, so here's the answer. If you remember, when we saved the file we had the computer store it in the file '.D1/SYSTEM.WRK-.TEXT'. As I said, it is a very important file. When you try to compile a program, the Compiler looks on the disk in the boot (internal) drive for a file with the name 'SYSTEM.WRK.TEXT'. If found, it compiles that file. If you are not using the workfile, it will ask you what file to compile. We will later learn how to save the file somewhere other than the workfile, but for now it makes things a little easier and quicker so we will use the workfile.

If you didn't have any errors in your program, and it compiled without a hitch, you can press the 'R' key to Run your program. Under the prompt line will be the line 'Running...', and under that the program will write the value of 821 + 120 (1001). Congratulations, you just wrote, compiled and executed your first Pascal program!

Press the 'F' key to enter the Filer and then press the 'L' key to list the files on a disk. When the prompt appears, enter '.D1' and then press 'RETURN'. This command will list all the files on the diskette in drive #1. In that listing you will see the files 'SYSTEM.WRK.TEXT' and 'SYSTEM.WRK.CODE'. The first you should recognize as the file with the lines you wrote and then compiled. What is the second one, though?

If you look back, you will remember that we compiled the file 'SYSTEM.WKR.TEXT'. Since the computer can not execute that file, the Compiler made another file - the 'object code' of the first file. This is the file that was Run. The word 'code' is the key! The file with the program lines that you typed in has the suffix '.TEXT'. This is the text file. The file that the Compiler created had a suffix of '.CODE'. This is the code file that was created by the Compiler and later executed with the Run command.

Say that you no longer want to add 821 and 180, but you want to multiply 27 by 8. We obviously need to change the program, so type 'Q' to exit the Filer, and the press 'E' to enter the Editor. If you remember the last time we used the Editor, it asked us for the file to use. Now, since there is a workfile, the Editor automatically reads in the text from the file 'SYSTEM.WRK.TEXT'. After a few seconds you should see the Editor prompt line and be ready to change the program.

While in the Editor, you can use the four cursor control keys to move anywhere within the file that you are currently working on. Use those keys to position the cursor over the plus '+' sign. Now press the 'D' key once. If you look at the prompt line, it now says '>Delete' and some more words that indicate what you can do with this option. Press the space key once and the plus sign will disappear. To make the change permanent, press 'CONTROL C'. Since we want to multiply and not add, press the 'I' key to insert more text and then type '*' and press 'CONTROL C'.

We just changed the addition to a multiplication, but we need to fix the numbers. We can now learn the Jump command to speed the change. The cursor should be somewhere in the middle of the lines on the screen. To move it to the beginning of the file, press the 'J' key. A new prompt line will appear. Press the 'B' key and the cursor will immediately jump to the beginning of the file. You can also jump to the end of a file by pressing 'J' and then 'E' for end. It doesn't matter if you type the letters in lowercase or uppercase letters, the computer understands what you mean.

Since we changed the program from addition to multiplication, we may as well change the name of the program. Use the 'J' key to move the cursor to the beginning of the file and press the space bar a few times to position the cursor over the 'A' in 'Addition'. Hit the 'D' key to enter the delete mode and then press the space bar eight times. Now press 'CONTROL C' to make the changes permanent. Finally, press the 'I' key to insert new text and then enter

'Multiplication' and 'CONTROL C' to finish up the insertion.

Before we got sidetracked with the Jump command, we were going to change the number 821 to 27 and the number 180 to 7. We can use the Replace option of the Editor to accomplish this. With the cursor at the beginning of the file, press 'R' and then enter '/821//27/'. The Editor will then find the characters '821' and replace them with '27'. Likewise you can change the number 180 to 7. This command is described in a little more detail on pages 130 to 135.

We could have used the Delete and Insert commands to perform this change, but it is very important to know many different methods of doing things. There are a number of other helpful commands that we will get to know, but for now the few that we have mentioned will be more than enough to get us through our beginning programming attempts. As I said, the Editor can double as a fine word processor and if you have ever used a word processor you should see what I mean.

Since we have changed our program, we can now save it back on the disk and leave the Editor. Press 'Q' to exit the Editor, and then 'U' to Update the work file. After some disk activity, press 'E' to exit the Editor. Now press the 'R' key to run the program and... Hey wait a minute, we still get 1001 as an answer! What went wrong? Oops, we forgot to compile the program! Because we didn't compile the updated program, the computer executed the old 'SYSTEM.WRK-.CODE' when we hit 'R'.

To compile the updated program, press the 'C' key to begin. Since we are still using the workfile, the Compiler didn't ask us for a file to compile - it just used the workfile. After a few seconds the main level command prompt line will come up again and the compilation will be done. Now press the 'R' key and you will be greeted with 216, which is the result of the multiplication 27 * 8.

One of the most important things in learning something new is not being afraid to make mistakes. The more mistakes you make at the start, the quicker you will be able to correct them down the road a few months. Therefore, lets put an error in our program and learn how to fix it.

At the main command level, type 'E' to enter the Editor. Since we are going to make an error on purpose, let's change the first word 'PROGRAM' to 'PROGRAN'. Position the cursor over the 'M' in 'PROGRAM' and hit the 'D' key once. Now press the space bar once to delete the letter, and then 'CONTROL C'. Next Insert the letter 'N' where the 'M' was. To finish, press 'Q' and then 'U' to quit and Update the file.

At the main command level hit 'C' to compile this new program. Ouch! What happened? The Compiler stopped with a weird message. Something about an error in one of the lines. To see exactly what error and where it occured, press 'E' to go to the Editor. After a few moments the computer will display the error message 'Error in declaration part. Type <space>'. Once you press the space bar, this handy little feature returns the cursor over the offending part of the program. In our case it is at the end of 'PROGRAN'.

When you think about it, this Pascal system is very nice because it can tell you not just where your errors are, but what type of error it was! This is very useful in debugging long and complex programs.

If you have been following along, you know that you have had to wade through a few pages of things that aren't terribly exciting. After all, you did start reading this column with the impression that you were going to learn Pascal, right? Well, the Pascal system comprises three disks of programs and hundreds of pages of reading. I think the keyword here is 'system'. Pascal isn't just a language, it's an idea - a feeling - a lot to learn!

Before we can really start in on the language, we must do the backround work. We have to learn about the Filer, the Editor, the Compiler and other features of the system, or we will never be able to figure out what Pascal is all about. Therefore, I think that if you want to learn Pascal you have to be very, very patient. You'll learn Pascal, but it isn't going to happen overnight.

At this point in time, if you have been following my lead, you should have a good idea about what the Pascal system is all about. You have used the Filer to create, modify and delete files. You used the Editor to make the source text for your Pascal programs, and you used the Compiler to compile that source text into a form the computer can understand. You have done all of the basic operations necessary to use the Pascal system, so what are we waiting for? Let's learn Pascal!

First Steps

If you've been following along, up to this point we have been reading out of the 'Introduction, Filer, and Editor' manual. Now we are going to take that big step and look into the 'Programmer's Manual Volume 1'. If you open up that book and look at pages 2 through 6, you will see a fairly good introduction to the Pascal language.

As it says on page 3, Pascal programs doesn't have line numbers. This free-form method of program design allows a much more readable program than in Basic or Fortran. This and other features of Pascal make programs much easier to write and maintain than equivalent programs written in other languages.

Page 5 shows the general structure of a Pascal program. Just as is shown, all Pascal programs have a number of parts. To start a Pascal program, the word 'PROGRAM' must come at the beginning of the file, followed by the name you want to program to be called and a semicolon. Note that this is not the same as the name of the file stored on the disk.

Some of the optional items that also can be included are the declarations of all variables and data types. If you don't know what these words mean, don't worry - I'll tell you in a little while. Next comes function and procedure definitions. The last part of a Pascal program is the word 'BEGIN' followed by any number of statements that are separated by semicolons, and the word 'END', followed by a period.

If you look back at the two programs which we already made, after the word 'PROGRAM' we put the names 'Addition-Test' and 'Multiplication-Test'. This is the program heading. Next in our programs came the word 'BEGIN'. This tells the compiler that the lines following this word are the main part of the Pascal program. In the PROGRAM 'Addition-Test', the next lines read 'WRITELN (821 +

180);'. This is a Pascal statement that prints out to the screen the result of whatever is inside its parenthesis. In our example, the computer added the numbers 821 and 180 and then printed out the sum to the screen. The exact meaning of 'WRITELN' is it instructs the computer to Write a LiNe to the screen.

Enter the Filer by pressing the 'F' key and then hit the 'N' key. After a second or two, when the new prompt line comes up, press 'Y'. If you remember, we have been working with the 'Workfile'. Whenever we enter the Editor it checks for a workfile. If it finds one, it loads it from the disk whether we wanted to use it or not. Since we now want to write another program we can use the 'N' command of the Filer to remove the workfile from the disk. You should not use the Remove command to do this because the New command updates some pointers in memory to say that there isn't a workfile anymore. Since the Remove command doesn't - don't use it.

Anyway, after you press 'Y' to confirm the removal of the workfile, the disk will make a little noise and the workfiles will be gone. Now Quit the Filer and press 'E' to begin editing a new program. Because there is no workfile, the Editor will now prompt you for a file to load. Just press 'RETURN'. Enter the program below, remembering to type a semicolon after all the 'WRITE' and the first 'WRITELN' statement.

```
PROGRAM Text_Test;
BEGIN
  WRITE ('This is the first part of the line, ');
  WRITE ('this is the second, ');
  WRITELN ('and this is the last.');
  WRITELN ('This is another line.')
END.
```

After you finish, remember to press 'CONTROL C' to accept the lines you just typed. Now hit 'Q' to exit the Editor and then 'U' to Update the workfile. Next, compile the program and press 'R' to execute the file. You should be greeted with 'This is the first part of the line, this is the second, and this is the last.' on one line, and on the next line will be the words 'This is another line.'.

You just learned three new things about Pascal. In the 'WRITELN' statements we used before, there were only numbers. Now you can see that it can display textual information also. To get the 'WRITELN' statement to print out text and not numbers, enclose what you want printed within single quote ' marks.

Isn't that great? We can print out text in additon to numbers. But what is that 'WRITE' statement? If you remember the meaning of 'WRITELN' you should see that they are quite similar. 'WRITE' prints out the information within its parenthesis to the screen, but doesn't drop on down to the next line when it is done. This explains why the first three Write statements print out only one line of text on the screen.

To test our fledgling knowledge of Pascal, let's make a couple of intentional errors. Enter the Editor and delete the semicolon after the first 'WRITE' statement. Now Quit the Editor and Update the file. Compile the program and in the middle of the compilation you will get an error message. Press 'E' to enter the Editor to see the mistake. In a few moments the line "Illegal symbol (maybe missing or extra ';' on line above)"will appear. Press the space bar to continue. The cursor will be right after the second 'WRITE'.

See how nice the system is? It tells you almost exactly what you did wrong so you can correct it. This is a whole lot better than some language compilers, whose short error messages are undecipherable.

A semicolon must always separate two consecutive statements. Since we deleted one in the above program, that gave us an error, you should be able to see why we got the error. But why aren't there semicolons between the 'BEGIN' and the first 'WRITE', and the last 'WRITELN' and the 'END.'? How can we justify this apparent contradiction? You probably see the answer. 'BEGIN' and 'END' are not statements!

You should think of these non-statements as nothing more than the separators of different parts of the program. Statements within the separators need to be separated by semicolons, but semicolons are not needed to separate the statements from the separators.

This is one of the most important things we will learn. Since Pascal is block structured, and in each program there will be many small program pieces, there are going to be a lot of 'BEGIN's and 'END's. Though there are many of them, the are NOT statements.

Get into the Filer and use the 'N' command to delete our workfile. Now Quit the Filer and enter the Editor. Type in the following program, Compile and run it.

```
PROGRAM Compact; BEGIN WRITELN ('Wow, it works!') END.
```

One of the nice features of the Pascal compiler is that it allows you to write your program in a free-form manner that is easy to read. As you can see the above program works just fine. In all of the previous programs, the extra lines and spaces were added only to make it more readable. The compiler can tell what you mean if you follow the few simple rules.

There is no set way of adding spaces or formatting your program, some people add lines and lines of extra indentation etc. to make their programs as easily understood as possible. I can't force you to add those spaces, but think of it this way. If you just finished writing a very long program and you had to debug (fix errors) it, which program would you rather read, the one with no indentation, or the one that has many extra spaces?

Congratulations, by now you have reached an important level of understanding Apple /// Pascal. The thirty or so single letter commands that you have learned are the ones that you will use most of the time in the future. In the past few pages you used them to Edit, Compile, Run and change your first Pascal programs. In the coming months, the programs you type in will get a little longer and more complicated, but you will be doing the same things that we did today - Edit, Compile, Run and change your programs.

It's about time to pack it in. You learned a lot about the Pascal system today. Next time we will learn some more of the commands of Pascal. If you like to work ahead, you should type in and compile the program on page 6 of the Programmers manual, and read through chapters 2 and 3. If you have any problems with this lesson, I will answer letters, so write me in care of ON THREE and keep on trying!

REVIEW ON: Apple Writer ///

by Bob Consorti

Managing written information, that is what word processing is all about, and that is what Apple Writer /// does. How easy does it work? Are there any bugs in the system? These are the questions people ask, and these are the questions that I'm going to try to answer.

Apple Writer ///

This program is one in a series of word processing programs written by Paul Lutus. He wrote the popular Apple Writer][for the Apple][, and this version is an extension of that program. However it's not just an Apple][copy (as many Apple /// programs are), but a completely new version that has some unbelievable features.

As all word processors do, with Apple Writer /// you type information in on the keyboard and it is stored in the computers memory. Once you have it in the computer you can do all sorts of neat things to the text. Since this aspect of word processing is very similar for all computers I'm not going to go into the detail of how each different function works. Instead I will tell of the features which makes this program different from the rest.

This program requires a 128K Apple /// and just the internal disk drive to work. A printer and extra disk drive are optional but come in handy. One of the programs weak points is that if you have a 256K computer, you still can only use about 64K of memory. In other words, don't go out and spend hundreds of dollars on a memory upgrade and expect Apple Writer /// to use the extra memory. I'm told that this will NOT be corrected in the future, so it may affect your purchase of this program.

There are three program disks in the package. It comes with two copy protected boot disks and one copyable utility disk. As is standard with Apple Special Delivery Software, they come with a 90 day warranty. The copy protection scheme used discourages multiple writings to the disk, so try to get your system configuration right the first time. After you write a new SOS.DRIVER file on the disk a few times, it refuses to boot and gives you an I/O error.

While we are on the subject of shortcomings, lets turn to the instruction manual for a moment. While fairly thorough and well written, the darn thing doesn't have an index! Any time you need to look up one of the many features of the program, you have to search through the Table of Contents. Since it is five pages long, you are in for a lot of searching. Even though they try to make up for no index with a tear out reference card, it isn't the same thing! I don't know why Apple allowed this, and as far as I'm concerned it is unexcusable! The whole reason for computers is to make our lives a little easier, and this type of instruction manual does not help - it hinders.

The program has a variety of very nice features. As with most programs of this type, there is a 'Data' line at the top of the screen. Here you can find information on how much memory is left, how much is used, the current position of the cursor in the file and the pathname of the file you are working on. Unlike many other programs, Apple Writer /// allows you to 'toggle' this line on and off. Thus, if you get tired of looking at it, one press of the 'ESCAPE' key will erase it and another will restore it!

If you are new to word processing, there is a thing called 'Wrap-Around'. This feature allows you to type information into the computer and as the words reach the right side of the screen, if they won't fit on the line, the whole word is brought down to the next line. Thus you only need to insert a Carriage Return at the end of a paragraph. Apple Writer /// allows you to toggle this feature also.

Another nice touch is the ability to 'see' these Carriage Returns. This is particularly helpful in the situation when you get a printout that you didn't expect. Sometimes you accidentally type in a Return and you don't know it. This feature allows you to see where all the Returns are in your text.

When typing in text, one feature that I don't like is where the cursor is. Instead on being over the character, it goes between them, splitting up words as it goes. Just a small point that I don't care for.

You can have a split screen display for working on one part of your file while displaying another. This is a very handy and sophisticated tool. Instead of relying on your memory when retyping in a common phrase, you can use the split screen and see the other portion of the file!

Apple Writer /// has the standard character and word delete, but the way that it implements these functions leaves something to be desired. Some programs use the numeric keypad as a special functions area. One key to delete a word, another restore a word, etc. This word processor uses 'CONTROL W' to delete and restore a word. Since the Apple /// makes it possible for the programmer to make an extraordinary program, I sometimes wonder why many programmers don't fully use the abilities of the machine.

You can toggle the word 'Wrap-Around' by pressing 'CONTROL Z'. This is another handy feature not normally found on word processors at this level. Perhaps the single best (well, one of a few) features of this program is the built-in Help screens. These provide a summary of most all the features found in the language. By pressing Open Apple - Question mark, a menu appears. Simply type in the number of the command you want explained, and in a moment it is displayed on the screen. It is so very helpful, that many of the newer Apple /// programs are incorporating it.

Another of the remarkable things that Apple Writer /// does is the Load and Save command. Like other word processors, you can save and load a file to and from a disk. Unlike others, this program allows you to save portions of the file in memory to the disk, load a particular segment of the file on the disk, append a file in memory to a disk file and more!

You can search a disk file for a particular phrase or string, and load it selectively into memory. This command lets you even load and save a file from the text in memory. Thus you can repeat sections of text very easily. You can also preview the contents of a disk file using the load command. When used, the text in the disk file is displayed on the screen but not stored in memory. The instruction

manual shows you how to access all of these features with good examples.

Another of the features of Apple Writer /// is the SOS Commands Menu. Invoked by typing 'CONTROL O', you can Catalog the files on a disk drive, rename, lock, unlock, and delete files. With this menu you can also set the Prefix so you only have to type in a local pathname, and even look at and set the current date and time.

The Catalog command has a slight problem that may give you a headache or two. It will only tell you the number of bytes in the file MOD 65536. Thus, if you have a very big file (> blocks), it will give you the wrong End Of File.

The 'TAB' key is fully functional and will go to every eighth column position when pressed. The user can very easily set a new Tab stop, delete an old one, or purge all Tab stops. You can also save the new tab settings to disk to be used later.

Apple Writer /// allows the user to set up a glossary and later invoke the glossary entries as macros. Simply put, you can tell the computer that whenever you type 'CONTROL Ga', the words 'Apple Writer ///' will appear. This allows you to write words or phrases that are difficult to type just once. From then on, only two key strokes will bring up any of the phrases you defined.

There is even a feature that allows you to change the case of any word or phrase without having to retype it. Just press 'CONTROL C' and then move the cursor over the word or character to be changed, and it will change case.

One of the most interesting features is the ability to change the style of font that is printed on the text screen. There are four extra fonts on the boot disk and you can change to any one of them at any time. If you like your characters to have a slant, be gothic looking, or have any other style, Apple Writer /// can give it to you.

The Find command lets you search through the text in memory for a particular word or phrase and also lets you change it. Very powerful, you can use this command with special delimiters to search for Returns, and even use wildcards to search for any sequence of characters.

Underlining, super and subscripts and footnotes are all supported, though some are a little hard to use. For instance, the underlining command forces you to replace any spaces within the text you want underlined to the underscore character. This is time consuming and is a very good example of what needs to be changed with this program.

Since one of the most important parts of word processing is printing the text file to a printer, we will now discuss the strong and weak points of Apple Writer ///'s Print commands.

To get into the 'Print Mode' of Apple Writer ///, press 'CONTROL P' and then enter a question mark. Displayed now is a list of the present printing format values. You can change the margins, specify how many lines to be printed per page, and even have page numbers and titles be printed on each page.

You can set the number of blank lines to be printed between each printed lines. Thus single, double and triple spacing are very easy.

You can also change the device to which the text will be printed. However, if you send the text to the .CONSOLE, you will not be able to see the text as it will appear on the printed page because it does not have a left-right scrolling ability. This is a serious deficiency and should be corrected.

At times you may want your printed text to be justified in a particular manner. You can force a left, center or right justification - but the fill justification feature has a very serious problem which makes it unworkable.

If you specify fill justification, spaces are added to the line, to line up both the right and left margins so they are flush. However, each line is fill justified they same way. Starting at the left margin, spaces are added to the line to fill justify it. This is a serious error because when you look at the printed page, there is more spacing at the left than at the right, and you get a section of text that is 'right heavy'.

Second year computer science majors are taught that this is an unacceptable practice and are shown how to correct it. It is really very simple. When you need to fill justify a number of lines, choose a random starting point within the line where to start adding spaces. This method gives a truly random look to the spaces in printed text and is the standard method for formatting during a fill justify. I don't know why this program doesn't have this feature, and until it does I would suggest that you don't use the fill justify mode.

While the print commands of Apple Writer /// are for the most part standard, it does allow you to paginate your documents by adding titles, page numbers etc. However it does have some problems that should be fixed.

The last of Apple Writer ///'s features that we will discuss is WPL, or Word Processing Language. This is the most revolutionary feature of Apple Writer ///. What is it? Just a language within a language, essentially a Basic interpreter that allows you to automate many of the repetitious chores associated with word processing.

You can use it to automatically print form letters, each with a different name and address that is stored in another file and is called up by WPL and inserted in the right places in your document.

WPL will translate one series of statements to another quickly and easily. Persons who use typewriter shorthand can use WPL to translate it back into recognizable sentences without operator intervention.

You can use WPL in a variety of tasks to do anything that you would have done by hand - only faster, automatically and it doesn't make mistakes!

Since WPL is really a computer language, ON THREE will periodically publish useful WPL programs and instructions on how to use them. We had hoped to have a special on WPL ready for this issue, but as usual we got behind and it didn't make the deadline. We will publish it next issue because it is a good introduction to Word Processing Language and tells just what makes WPL so revolutionary. If you have written a particularly interesting program in WPL I want to hear from you!

Utilities Diskette

One of the diskettes included with the Apple Writer /// package is the Utilities disk. This contains a utility program which allows you to transfer Apple Writer][files to Apple Writer /// files and vice-versa. You can also use this program to transfer Mail List Manager files to a form that can be used by Apple Writer ///. Once the Mail List Manager files are in the format for Apple Writer /// files, they can be used in form letters and other things that Apple Writer /// and WPL can do.

The program is written in Pascal and if you have Pascal you can execute it as any other Pascal code file. If you want to do this, you must first copy the dependent Units from the Utilities Diskette 'SYSTEM.LIBRARY' to your own 'SYSTEM.LIBRARY' for it to work.

You must have one external drive for the program to work. The Apple Writer /// operating manual has instructions for using the Utilities diskette in Appendix C and D. If you use Apple Writer][and Apple Writer /// files, this utility is very handy.

Product Training Pak

The Apple Writer /// Product Training Pak is a tutorial manual and diskette that will introduce you to the Apple Writer /// word processor. If you are new to computerized word processing this is a very helpful package to use to learn Apple Writer ///. If you are using your computer in your office and you want to teach your secretary how to use it, this manual is invaluable.

Starting from the very simplest of commands, the tutorial teaches the novice user most all of the functions that are required to operate the word processor. Loading, and Saving of files are all covered, in addition to how to use the Help screens.

Simple character insertion, deletion and replacement are covered. The manual also tells how to access the SOS commands menu and how to use it. Basic printing operations are also covered.

Even WPL is given a few pages. The demonstration disk that comes with the Product Training Pak contains many example WPL programs that the tutorial shows how to use. A form letter filling program automatically writes out letter after letter using a mailing list on another one of the disk files. WPL printing of files is also shown, as well as a program that changes fonts.

Also on the disk are a number of very interesting files. They were originally deleted, but with a little expertise you can restore them (see 'Lazarus ///' in a future issue). One of them is the text file of the Product Training Pak instruction manual! You will be able to see the instruction manual on your screen!

Since the Product Training Pak is a very useful addition for novice users, I would recommend purchasing it when you buy the main program. If you have any fears about word processing, it will do a lot to alleviate them.

Summary

Apple Writer /// is certainly a good word processor, but should you purchase it for your computer? Well, that's your decision, but let me say that if Apple Writer /// didn't have WPL I would recommend that you didn't buy it. There are quite a few things that should be changed in the program before it is one of the quality items that Apple /// users deserve. However, because Apple Writer /// has WPL, the good points outweigh the bad points and I think it is a good word processor.

Equipment used in this review:

128K Apple ///
1 external floppy drive

Program: Apple Writer /// for the Apple ///
Version: Interpreter created 9/18/81
Contents: Program, Backup and Utilities Diskettes.
User's manual.
Programming language: Assembly and WPL
Operating System: Standard SOS
Copy Protected: Yes
Disk Warranty: 90 days
Backup disk included: Yes
Cost: $225.00

Program: Apple Writer /// Utilities
Programming language: Pascal
Operating System: Standard SOS
Copy Protected: No
Cost: Included with Apple Writer ///

Product Training Pak:
Contents: Tutorial manual
Sample Data Files Diskette.
Cost: $40.00

The Bottom Line

Apple Writer ///

Performance: Good
Documentation: Poor-Fair
Ease of use: Fair-Good
Error Handling: Good
Over All Rating: B-

```
    New_page ('Press any key to continue')
  END; { Of PROCEDURE Show_disk_info }

  PROCEDURE Do_it;        { Performs the listing of the directory }
  BEGIN
    Get_root_info;        { The root block has 12 files positions }
    Show_root_info;       { plus a root volume name on it. }
    Get_file_info (12);
    IF ((Out_Path <> '.CONSOLE') AND (Out_Path <> '#1')) THEN
      WRITE ('Writing.');
    Show_file_info (12);
    WHILE (NOT EOF (Infile)) DO
      BEGIN
{$IOCHECK- }
        Count := BLOCKREAD (Infile, Block_buf, 1);
{$IOCHECK+ }
        Trap_IO_error;
        Get_file_info (13);     { All other directory blocks have }
        Show_file_info (13)     { 13 file positions available.    }
      END;
    Show_disk_info
  END; { Of PROCEDURE Do_it }

BEGIN { MAIN of List_SOS_Directory }
  Set_Error_types;
  Set_File_types;
  Set_Out_device;
  File_count := 0;
  Line_count := 0;
  Open_Directory;
  Do_it;
{$IOCHECK- }
  CLOSE (Infile);
  CLOSE (Device, LOCK);
{$IOCHECK+ }
  Trap_IO_error
END; { Of PROCEDURE List_SOS_Directory }

BEGIN
  { This is the initialization, which occurs }
  { before the host program is executed. }
END. { Of UNIT List_Stuff }
```

Errata: Opps, a mistrake!

If you purchased the January Disk of the Month, you are no doubt wondering why the program 'FONTDEMO' on the back side of the disk doesn't work. To fix it make the following changes to the program.

```
20    a$="a%":i=1
110    IF MID$(b$,3,4)<>"FONT" THEN 10
       0:ELSE a$(i)=b$:n$=MID$(b$,16,1
       5):FOR j=1 TO 15:IF MID$(n$,j,1
       )<>" " THEN NEXT
340    IF k$<>" " THEN NEXT:ELSE END
```

Since you can't save the program back to the diskette (it has no write-protect notch), save the file on another disk. To use, transfer the subdirectory 'FONTS' and all the files in it to the other disk.

Three Shorts — Fini!

Once again, kick off your boots and soothe that aching head to the sights and sounds of ON THREE! The first two programs are graphic demos, while the last is a sound demo! It will give you a chorus of weird noises.

To use the program "Bob's Blocks", just type it and and enter "RUN". No other files are needed. For the program "Bob's Lines", you need to have the file "BGRAF.INV" on a disk named "/BASIC" in one of your drives. If this file is elsewhere, make the appropriate change to line #100. If the program can not find that file, it will hang, and you will have to press the RESET key to stop it.

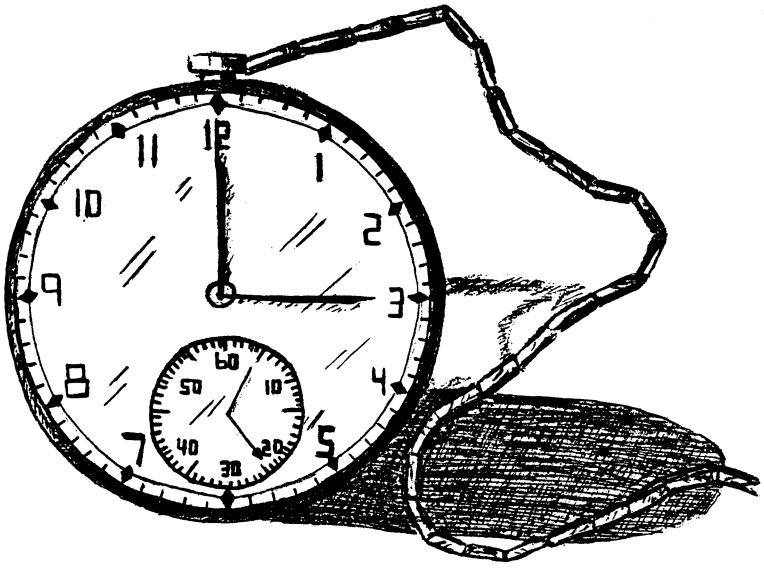
The program "Bob's Noises??" uses the ".AUDIO" driver to generate weird sounding noises. If the ".AUDIO" driver is not configured into your system, you will get an error message.

ON THREE will pay $25 for any short demonstration program used in this space, so send in your favorite today, and we will see you next time in ON THREE. ///

```
0   REM ******************************************
1   REM *  Bob's Blocks                          *
2   REM *  ------------                          *
3   REM *  This simple little program shows how *
4   REM *  fast the Apple /// can display diff-  *
5   REM *  erent colors on the screen.  Just     *
6   REM *  type in the program and enter 'RUN'.  *
7   REM *  This works best with a color monitor *
8   REM *  but is still okay without one.        *
9   REM ******************************************
10   x.left%=0:x.right%=0:y.top%=0:y.bot%=0
20   loop.color%=0:back.color%=20:clear.view%=28
30   black%=0:white%=15
40   text.mode%=16:color.mode%=1
50   PRINT CHR$(text.mode%);CHR$(color.mode%)
60   HOME
99   ON KBD GOTO 1000
100   FOR loop.color%=black% TO white%
110     GOSUB 200
120     PRINT CHR$(back.color%);CHR$(loop.color%);
130     PRINT CHR$(clear.view%);
140     NEXT loop.color%
150   GOTO 100
200   x.left%=RND(1)*40:x.right%=RND(1)*40
210   IF x.left%>x.right% THEN SWAP x.left%,x.right%
220   y.top%=RND(1)*24:y.bot%=RND(1)*24
230   IF y.top%>y.bot% THEN SWAP y.top%,y.bot%
240   WINDOW x.left%,y.top% TO x.right%,y.bot%
250   RETURN
1000   IF KBD=27 THEN POP:TEXT:HOME:END
1010   ON KBD GOTO 1000
1020   RETURN
```

```
0   REM ******************************************
1   REM *  Bob's Lines                           *
2   REM *  -----------                           *
3   REM *  Another very short program, it will   *
4   REM *  draw lines on your graphics screen.   *
5   REM *  It pickes random endpoints for the    *
6   REM *  lines so you will get a colorful      *
7   REM *  straw affect.  To use, make sure you  *
8   REM *  have '/BASIC/BGRAF.INV' on line.      *
9   REM ******************************************
100   ON ERR INVOKE"/BASIC/BGRAF.INV"
110   PERFORM initgrafix:OFF ERR
120   xdist%=140:ydist%=192:mode%=3:buf%=1
130   PERFORM grafixmode(%mode%,%buf%):PERFORM grafixon
140   PERFORM fillport
199   ON KBD GOTO 1000
200   FOR a%=0 TO 1000
210     PERFORM pencolor(%RND(1)*16)
220     PERFORM moveto(%RND(1)*xdist%,%RND(1)*ydist%)
230     PERFORM lineto(%RND(1)*xdist%,%RND(1)*ydist%)
240     NEXT
250   GOTO 200
1000   IF KBD=27 THEN POP:TEXT:HOME:END
1010   ON KBD GOTO 1000
1020   RETURN
```

```
0   REM ******************************************
1   REM *  Bob's Noises??                        *
2   REM *  --------------                        *
3   REM *  Up to now we have used this column to *
4   REM *  show off the graphics capabilities of *
5   REM *  the ///.  This program shows you some *
6   REM *  of the different sounds the Apple /// *
7   REM *  can make.  Just type it in and enter  *
8   REM *  the command 'RUN'.                    *
9   REM ******************************************
100   PRINT CHR$(14);:REM Turn off the screen
110   count%=0:time%=1:c.val%=16383:mode%=128:vol%=63
120   OPEN#1,".audio":GOTO 499
200   PRINT#1;CHR$(mode%);CHR$(vol%);
210   PRINT#1;CHR$(count%-256*INT(count%/256));
220   PRINT#1;CHR$(INT(count%/256));
230   PRINT#1;CHR$(time%-256*INT(time%/256));
240   PRINT#1;CHR$(INT(time%/256));
250   RETURN
499   ON KBD GOTO 1000
500   FOR a%=1 TO 1000
510     count%=INT(RND(1)*c.val%)
520     GOSUB 200
530     NEXT
540   GOTO 500
1000   IF KBD=27 THEN CLOSE#1:TEXT:HOME:END
1010   ON KBD GOTO 1000
1020   RETURN
```



Disk Of the Month

Calling all you busy professionals, would you like to have the programs in this month's issue? What's that? You don't have time to type them in yourself? Well, just buy the disk!

This disk contains all the programs contained in the January and the February-March issues of ON THREE. Included are Disk Pak1:, which will give you extra disk space; Disk Pak2, which lists the files on a directory using Pascal; both of the Key-Things programs; all of the Graphics and Sound Demos and more!

To discourage piracy, we have priced these disks so low, that everyone can afford one.

Buy one now for the low, low price of $9.95 (Plus $1.50 for postage and handling).

Group rates are as follows:

2 - 9 disks: **$7.50** apiece + $3 total shipping
10-24 disks: **$7.00** apiece + $4 total shipping
over 24 disks: **$6.50** apiece + $5 total shipping

Group rates must have one mailing address. Please use the attached envelope for orders. If envelope is missing, send to:

ON THREE
Attn: ORDER DEPT.
P.O. Box 3825
Ventura, California 93006

ON THREE O'Clock

How would you like a working clock/calendar for your Apple ///? Just as it was originally intended, a plug in clock chip with a battery backup.

With ON THREE O'Clock installed, any time you save or modify a file, the current date and time will be stored on disk. Thus you will now be able to tell which file you last worked on, etc.

Extremely easy to install and adjust, this is the one you have been waiting for!

This package contains comprehensive instructions and a Six Month Warranty! Try to get that deal anywhere else!

What's the best part? - The price! While others are selling theirs for $60 and up, we have broken the $50 barrier. Heck, we broke the $40 barrier!

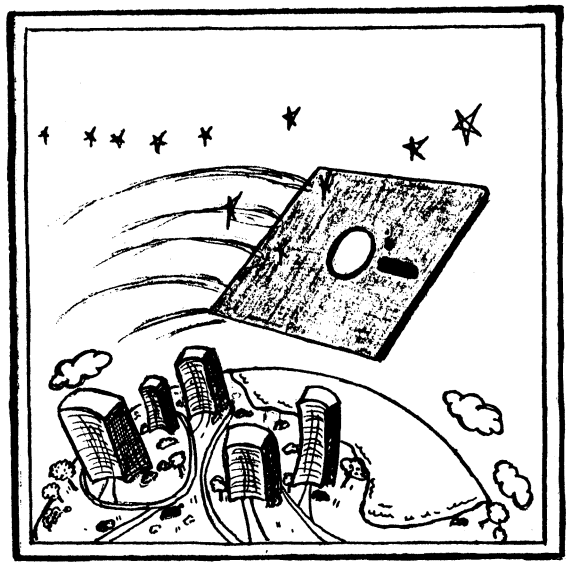
For only $39.95 (plus $2.50 for postage and handling) you can get the best little clock in town!

Group rates are as follows:

2 - 9 clock sets: **$36.50** apiece + $5 total shipping
10 - 24 clock sets: **$33.25** apiece + $7 total shipping
over 24 clock sets: **$31.00** apiece + $9 total shipping

Group rates must have one mailing address. Please use the attached envelope for orders. If envelope is missing, send to:

ON THREE
Attn: ORDER DEPT.
P.O. Box 3825
Ventura, California 93006



ON THREE
P.O. BOX 3825
VENTURA, CA 93006